UNIVERSITE DES SCIENCES DE MONTPELLIER — ECOLE DOCTORALE SIBAGHE

GENERATION SEMI-AUTOMATIQUE DE SERVICES WEB SEMANTIQUES POUR DES BASES DE DONNEES RELATIONNELLES BIOLOGIQUES

THESE

présentée et soutenue publiquement le 13 décembre 2011

pour l'obtention du titre de :

Docteur de l'Université Montpellier 2

par

Julien WOLLBRETT

Jury

Rapporteurs :	Christine Froidevaux, Professeur, Université Paris-Sud 11 Hadi Quesneville, Directeur de Recherche, INRA, URGI
Examinateurs :	Patrice This, Directeur de Recherche, INRA, UMR AGAP Philippe Bessières, Directeur de Recherche, INRA, MIG
Directeur de Thèse : Co-Directeur deThèse :	Frédéric de Lamotte, Chargé de Recherche, INRA, UMR AGAP Manuel Ruiz, Chercheur, CIRAD, UMR AGAP
Invités :	Pierre Larmande, Ingénieur, IRD, UMR DIADE Isabelle Mougenot, Maître de conférences, Université Montpellier II

CENTRE INTERNATIONAL DE RECHERCHE AGRONOMIQUE POUR LE DEVELOPPEMENT – UMR AGAP

Remerciements

La réalisation de mon projet de thèse n’aurait pu être possible sans l’aide et le soutien de plusieurs personnes auxquelles je tiens à manifester toute ma gratitude.

Je souhaite remercier en premier lieu mon directeur de thèse Frédéric de Lamotte pour le temps qu’il m’a accordé ainsi que pour ses qualités pédagogiques et scientifiques.

Je remercie également Manuel Ruiz, mon co-directeur de thèse ainsi que Pierre Larmande qui m’a encadré. Ils m’ont tous les deux suivis de prêt durant ces 3 dernières années et je n’aurais pas réussi à réaliser ce travail sans leur aide.

Je voudrais également remercier Isabelle Mougenot, qui m’a accordé beaucoup de temps et qui m’a donné des conseils précieux, me permettant de sortir de nombreuses impasses.

Je remercie aussi Christine Froidevaux et Hadi Quesneville, qui ont eu la gentillesse d’accepter d’être mes rapporteurs et qui ont puisé dans leur temps de recherche pour lire mon mémoire. Leurs critiques constructives m’ont permis de prendre du recule par rapport à mon travail.

Je remercie Philippe Bessières et Patrice This, d’avoir accepté d’être examinateurs de mon travail.

J’associe à ces remerciements tous les membres de l’équipe Integration des Données : Gaetan, Stéphanie, Xa, Jean François, Chantal, Bertrand, Marilyne, Guilhem, Gautier, Manon et ma chère voisine de bureau Cécile.

Je remercie tous les membres du Batiment 3 pour leur bonne humeur et leurs débats lors des pauses café, ainsi que toutes les autres personnes qui ont participé indirectement à cette aventure.

Je tiens à témoigner ma gratitude à ma famille pour son soutien au cours de ces longues années d'études, sans eux je n'en serai pas là aujourd'hui. Je remercie plus particulièrement ma maman qui m’a aidé à financer mes études et qui a toujours su trouver les mots justes pour me remotiver.

Enfin, je désire remercier Lucile pour son soutien moral, son amour et sa patience, même s’il faut avouer que j’ai toujours été très facile à vivre…….

À mon papa qui nous a quitté trop tôt,

Sommaire

Introduction

Contexte biologique

Un des apports majeurs de la génomique végétale est une compréhension accrue des évènements de domestication et d'adaptation des plantes cultivées, et notamment des plantes cruciales pour l'alimentation des populations du Sud (riz, sorgho, bananier, manioc, etc.). Les projets de génomique végétale intègrent de plus en plus les nouvelles technologies de séquençage à très haut débit (Next Generation Sequencing ou NGS). Ces technologies révolutionnent non seulement les approches expérimentales en biologie, mais aussi le traitement bioinformatique de tels volumes de données qui exige le développement de nouvelles méthodes d'analyse suffisamment performantes pour passer à l'échelle. L'application des NGS ne se limite pas au séquençage de nouveaux génomes, mais touche aussi le reséquençage de génomes, l'identification de variations génomiques telles que les SNPs, ou la transcriptomique.

En parallèle, le développement des méthodes de phénotypage à haut-débit engendre aussi une masse d'information considérable, qui couplées aux données de séquençage, permettent notamment des études d'association à l'échelle des génomes (X. Huang et al. 2010).

En outre, les chercheurs, qui travaillent dans le domaine de la diversité, de la génétique, et de la création variétale ont aussi besoin de connecter cette masse de données provenant d'expériences de génomique à haut-débit avec des données provenant de bases de données de ressources génétiques. Les informations relatives aux accessions (ou lot, ou écotype) conservées dans les centres de ressources génétiques sont très diverses : (*i*) des données de passeport qui incluent les noms scientifiques et vernaculaires, les synonymes, une localisation géographique, éventuellement des caractéristiques de milieu, les types d'usage, le pédigrée, (*ii*) des données de caractérisation, en général sur des critères morphologiques, tels que définis par des instituts internationaux comme Bioversity[1], et (*iii*) des données de gestion interne: localisation, stock, disponibilité, contrôle et validation, commandes, historique des échanges…

Toutes ces données sont accompagnées de méta-informations caractérisant les données elles-mêmes et leur processus d'acquisition (par exemple paramètres biochimiques d'un marqueur moléculaire, conditions expérimentales d'une évaluation phénotypique, etc.), ce qui

[1] http://www.bioversityinternational.org/

justifie que les concepteurs de bases de données préfèrent en général organiser ces informations dans des bases de données dédiées.

Spécificités des données biologiques

Les données biologiques sont hétérogènes et de plus en plus nombreuses. Ces données sont produites par une diversité de méthodologies expérimentales (séquençage de l'ADN, puces d'expression, double-hybride chez la levure, spectrométrie de masse, phénotypage métabolique, etc.) et correspondent à des types de données différents (séquences nucléotidiques, identifiant d'un gène surexprimé dans une condition donnée, interactions protéine-protéine, modifications post-traductionnelles, phénotypes qualitatifs ou quantitatifs, etc.). Chaque type de données pourra être stocké dans un format de données qui lui est propre (FASTA, MAGE-XML, SMBL, etc.). De nouvelles technologies d'analyses biologiques sont mises en place fréquemment, nécessitant la création de nouvelles plateformes de traitement de données et de nouveaux formats de données, impliquant une grande **évolutivité des données biologiques**.

Les données biologiques varient entre elles en termes de **qualité**. Cette variation de qualité peut être due au type d'étude réalisée pour obtenir la donnée, mais elle peut également provenir de la manière dont les données brutes sont analysées et annotées par les bioinformaticiens et/ou les biologistes. Par exemple, les bases de données Swissprot et Trembl contiennent des séquences protéiques annotées. Les données contenues dans Swissprot sont issues d'annotations manuelles. Les données de Trembl sont, quand à elles, issues d'annotations automatiques n'ayant pas forcément subit d'étapes de vérifications manuelles.

La **quantité de données** stockées est de plus en plus grande (Zhong & Sternberg 2007) en raison des nouvelles technologies à haut débit comme les puces à ADN ou les nouvelles technologies de séquençage.

Spécificité des sources de données biologiques

Les sources de données biologiques ont la particularité d'être très **nombreuses**, et en **constante augmentation**. Le supplément annuel, *Database Issue*, du journal *Nucleic Acids Research* dédié aux bases de données en biologie moléculaire, en a répertorié 96 en 2001

(Baxevanis 2001), plus de 800 en 2007 (MY Galperin 2007), et plus de 1300 en 2011 (Michael Galperin & Cochrane 2011).

La nature distribuée et les domaines d'application très variés de la biologie favorisent la création de sources de données spécialisées plutôt que la centralisation des données dans de grands systèmes de stockage. Cela implique que chaque type de données peut être présent dans de multiples sources de données, parfois de manière **redondante**. Par exemple, il existe plus de 231 bases de données différentes sur les cascades biologiques (Carole Goble & Robert Stevens 2008).

Beaucoup de sources de données sont **volatiles**, disparaissant ou étant remplacées par d'autres sources de données fournissant des données identiques ou similaires mais présentées d'une autre manière ou obtenues à l'aide d'une autre approche. Par exemple, Merali et Giles (Merali & Giles 2005) rapportent que seulement 18% des bases de données existantes bénéficieront d'un support dans le futur.

Les sources de données biologiques possèdent une forte **hétérogénéité syntaxique**, liée à la diversité des modèles de données. Les données biologiques sont disponibles dans des formats non structurés (i.e. fichiers texte), semi-structurés (XML), ou structurés (relationnel, objet, etc.). Il est cependant intéressant de noter que si l'on se réfère aux sources de données référencées dans Nucleic Acids Research online Database Collection[2] , la majorité des données biologiques publiques sont stockées dans des systèmes de gestion de bases de données relationnelles. Cela n'est pas spécifique aux données biologiques, car 77,3% des données stockées sur le Web sont également contenues dans des bases de données relationnelles (Chang et al. 2004).

L'hétérogénéité du contenu des sources de données est aussi de nature **sémantique** et recouvre plusieurs aspects : (i) le degré de spécialisation de chaque base de données, chacune se focalisant sur un type d'objet biologique et ne représentant pas l'information avec le même niveau de détails, comme par exemple les bases dites généralistes versus spécialisées, (ii) la diversité des modes de désignation des concepts biologiques selon différents vocabulaires entraînant des conflits sémantiques, (iii) le problème des identifiants différents pour un même objet biologique.

Une grande quantité de sources de données biologiques fonctionnent de manière **autonome** (Hernandez & Kambhampati 2004). Cela signifie qu'elles sont libres de modifier leur schéma ou supprimer des données sans aucune notification publique. Elles peuvent également

[2] http://www.oxfordjournals.org/nar/database/a/

bloquer l'accès à la source pour raison de maintenance. De plus, elles ne sont pas forcément au courant que d'autres sources les référencent ou qu'elles sont utilisées dans des systèmes d'intégration de données.

Vers une intégration des sources

Cette répartition des données pose problème aux biologistes car, pour réaliser certaines études, ils doivent souvent regrouper les informations contenues dans ces sources indépendantes. C'est par exemple le cas lorsque le chercheur veut croiser des informations sur les marqueurs moléculaires avec des allèles d'intérêts pour différentes populations, selon des critères comme les zones géographiques, l'origine des variétés, leurs structures génomiques, etc. De la même manière, pour comprendre l'effet d'une mutation particulière dans une séquence nucléotidique sur le transcriptome, le proteome, le metabolome et le phénome, il est souvent nécessaire de récupérer des données dispersées issues de bases de données spécialisées dans ces différents domaines de connaissance.

Le partage de l'information sur le Web a permis aux biologistes d'accéder à de multiples sources. Cependant, l'accès à ces sources est indépendant ce qui limite le degré d'automatisation de la recherche d'information. Dans ce contexte des solutions intégrant le contenu de plusieurs sources de données ont vu le jour. Parmi les approches classiques d'intégration, la médiation offre une intégration forte, c'est-à-dire basée sur une structure intégrative.

La médiation est une approche virtuelle simulant une intégration physique des données par l'utilisation d'adaptateurs interrogeant directement les sources de données dans leurs emplacements d'origine. Un adaptateur doit être créé par chaque fournisseur de données de manière à rendre sa source de données disponible sur la plateforme. Ce travail nécessite la mise en correspondance entre le schéma du médiateur et le schéma de la base de données, sur laquelle la création des adaptateurs s'appuiera. Cependant, la création de ces adaptateurs est encore consommatrice de temps. De plus, ces adaptateurs seront spécifiques au médiateur pour lesquels ils ont été réalisés. Le travail et le temps nécessaire à leur création dû à leur spécificité vis à vis du médiateur peut rebuter les fournisseurs de données et ainsi limiter le nombre de sources accessibles via ce système.

Vers le Web Sémantique

Pouvoir représenter la connaissance humaine et pouvoir raisonner à travers elle en terme de structure symbolique est un problème de longue date. Pour cela, il faudrait pouvoir spécifier explicitement les connaissances implicites, c'est-à-dire attacher des métadonnées aux ressources existantes, permettant ainsi de les définir mais également définir les relations entre les différentes ressources. Une récente application de ce travail nommé le Web Sémantique (Berners-Lee et al. 2001) a pour but de fournir un Web compréhensible par des machines en organisant les données en fonction de leur sémantique. La notion de Web Sémantique a été introduite en 1998 par Tim Berners-Lee qui le définit comme « un Web de données, en quelque sorte comme une base de données globale » (Berners-Lee 1998) dont le but est transformer le web existant en « un ensemble d'applications connectées […] formant un Web de données basé sur une logique cohérente » (Berners-Lee 1998).

Le point commun de l'intégration de données et des technologies du Web Sémantique est de dépasser l'hétérogénéité sémantique de sources de données interconnectées. L'utilisation du Web Sémantique facilite la représentation de la sémantique des données et peut ainsi être utilisé pour faciliter l'interopérabilité ou l'intégration de données.

Objectifs de mes travaux

Les travaux introduits dans ce mémoire proposent d'automatiser la création d'adaptateurs sémantiques, facilitant ainsi l'intégration de sources de données biologiques. Plus précisément, nous nous sommes intéressés au développement d'une méthodologie permettant d'automatiser au maximum l'intégration de bases de données relationnelles biologiques dans une plateforme d'intégration de type médiation. Notre approche se veut flexible, générique et automatisée. Pour cela nous nous appuyons sur des standards du Web Sémantique et des Services Web[3] pour développer de manière automatisée des adaptateurs sémantiques de type Service Web Sémantique.

[3] Un Service Web est une entité indépendante pouvant être invoquée à travers le Web, quelle que soit l'application logicielle. Les Services Web fournissent une interface de programme qui s'affranchit des interfaces Web pour récupérer des informations, par ailleurs les descriptions sémantiques favorisent leur découverte et leur composition.

Plan du manuscrit

Ce mémoire est divisé en 3 grandes parties.

La première partie présente l'état de l'art décrivant les bases nécessaires à mon travail de thèse. Cette partie comprend 4 chapitres. Le chapitre 1 présente les difficultés liées à l'intégration de données biologiques avant de détailler les approches d'intégration existant en biologie. Il est présenté plus en détail les approches d'intégration fortes de type entrepôt et médiation ainsi que le développement d'adaptateurs dans une approche de médiation. Le Web Sémantique est détaillé dans le chapitre 2 qui décrit tous les standards du Web Sémantique ainsi que leurs avantages dans l'intégration de données biologiques. Le chapitre 3 est dédié à la description des Services Web et présente les technologies de base utilisées dans leurs créations ainsi que les limites inhérentes à leur utilisation. Un point est fait sur l'intérêt de l'utilisation des Services Web Sémantiques. Le chapitre 4 se place dans le contexte de la mise en correspondance entre le schéma d'une base de données relationnelle et des ontologies, avec une présentation des différentes approches existantes.

La deuxième grande partie présente le contexte dans lequel s'inscrit mon travail de thèse, les objectifs, et comment les approches existantes vont pouvoir être utilisées.

La troisième grande partie de ce mémoire se focalise sur les contributions de mes travaux. Le chapitre 6 présente l'approche mise en place pour enregistrer automatiquement une ontologie OWL dans l'ontologie de types de données BioMoby. Le chapitre 7 présente la plateforme d'intégration mise en place afin d'automatiser au maximum la création d'adaptateurs de type Services Web Sémantiques. Le chapitre 8 décrit le développement d'une méthode originale de création automatique de requêtes à intégrer dans nos Services Web Sémantiques.

Le mémoire se termine évidemment par les parties discussion et conclusion.

Etat de l’art

1. L'intégration de données

1.1. Introduction

La bioinformatique est une discipline basée sur des données variées correspondant à des champs de recherche biologiques différents et réparties dans de nombreuses sources. Ces sources variées doivent pouvoir être combinées afin d'obtenir une vision globale et de déduire de nouvelles connaissances (Pasquier 2008). L'avènement de l'internet a largement contribué à la diffusion des données biologiques. La quantité de données stockées dans des bases de données a augmenté de façon exponentielle lors des dix dernières années[4], faisant passer le nombre de bases de données disponibles de 96 (Baxevanis 2001) à plus de 1300 (Michael Galperin & Cochrane 2011) durant la même période. En plus d'être nombreuses, ces sources sont fortement hétérogènes. Face à ces chiffres et à l'hétérogénéité des sources, il est devenu impensable pour un biologiste de réaliser une intégration manuelle à grande échelle.

Quatre caractéristiques principales majeures sont à prendre en compte pour pouvoir intégrer des sources biologiques (Hernandez & Kambhampati 2004) : la variété des données, l'hétérogénéité de représentation, l'autonomie de ces sources ainsi que leur capacité de requêtage. La variété des données pose problème pour identifier formellement des relations entre les données, notamment lorsqu'elles sont issues de plusieurs champs de recherche. L'hétérogénéité de représentation entre sources conduit à des questions sur l'identification d'entités entre ces sources ainsi que sur la qualité, la redondance et la cohérence des données. Les sources autonomes peuvent modifier continuellement leur contenu. Dans ce cas, le seul moyen pour un système d'intégration d'être certain d'accéder aux dernières données est d'interroger directement la source. Les sources fournissent fréquemment une interface d'interrogation n'autorisant que certaines requêtes prédéfinies. Cela peut empêcher la création de requêtes pertinentes même si la source possède les données nécessaires.

[4] http://www.ebi.ac.uk/embl/Services/DBStats/

1.2. Les systèmes d'intégration de données

1.2.1. Etat de l'art des approches

Une grande variété de technologies et de techniques d'intégration de données ont été utilisées au cours des 15 dernières années (Carole Goble & Robert Stevens 2008). Elles varient au niveau (i) de l'architecture qu'elles adoptent, (ii) de leurs degrés d'automatisation, (iii) de point de vue suivant qu'il soit basé sur la source de données ou sur le système d'intégration. L'ordre de présentation des différentes approches est organisé ici en fonction de leur capacité d'intégration, de la plus légère à la plus complète (Figure 1). Une intégration forte fournit un schéma, un langage et une interface transparente alors qu'une intégration lâche n'offre que la transparence (Guérin 2005). Les deux approches d'intégration les plus complètes, à savoir les entrepôts de données et l'intégration par vue, seront plus spécifiquement détaillées par la suite.

- Les **applications composites** (mash-up) sont des applications récupérant des données issues de plusieurs sources (Belleau et al. 2008) de manière à les agréger et les afficher d'une façon différente. L'exemple le plus connu d'application composite est Google Maps qui permet de coupler des éléments venant de sources multiples en les cachant derrière une interface graphique simple. L'émergence de cette technologie a été alimentée par le développement d'applications utilisant la technologie AJAX et par le nombre d'API Web renvoyant des informations issues de sites web populaires comme Flickr, YouTube ou Google Maps (Thor et al. 2007). Actuellement, il existe plus de 6 000 (ProgrammableWeb.com 2010) applications composites utilisables pour créer des pages web. Les applications composites permettent d'agréger les données sans les intégrer réellement.
- **L'intégration navigationnelle** consiste à réaliser une référence croisée d'une donnée issue d'une source avec une autre donnée issue d'une autre source. Comme ces données sont habituellement présentées sous forme de pages HTML, l'utilisateur peut suivre ces références croisées à l'aide de liens hypertexte. Les systèmes comme SRS (Etzold et al. 1996), Entrez (Schuler et al. 1996) et Integr8 (Kersey et al. 2005), sont des portails et des systèmes d'index de mots clés qui maintiennent le réseau de liens entre ressources. Il s'agit de l'approche d'intégration la plus utilisée actuellement, et qui est supportée par la majorité des fournisseurs de données. Par exemple, SRS représente 40% du trafic de EMBL-EBI (Carole Goble & Robert Stevens 2008). Il

s'agit plus d'une approche permettant de relier des sources plutôt qu'une approche d'intégration. Cette approche est vulnérable à des conflits de noms et aux mises à jour.

- Les **applications d'intégration** telles que Ensembl (Hedeler et al. 2007) ou Ondex (Koehler et al. 2005) sont construites dans le but d'intégrer des données d'un domaine d'application particulier. La plupart de ces approches sont basées sur une architecture dirigée par modèle. Le point positif de ces applications est leur capacité à bien intégrer les données de leur domaine d'application. Elles sont difficiles à étendre.
- Les **flux de travaux** (workflow) sont très populaires (C. F. Taylor et al. 2007). Ils permettent de décrire et de lancer une série de processus reliés. Pour l'intégration de données, les flux de travaux coordonnent un flux de données entre des services de données et des outils d'analyses. Il existe des outils très puissants permettant de créer facilement des flux de travaux. C'est le cas de Taverna (Oinn et al. 2004), un logiciel open source qui permet de les créer graphiquement, en manipulant et en combinant des briques élémentaires (chacune réalisant une tache précise), puis de les exécuter facilement.
- L'**architecture dirigée par modèle** est une version d'intégration par vue très utilisée par les grands projets biologiques comme par exemple GMOD (Generic Model Organism Database project) (GMOD 2007) ou caBIG (Covitz et al. 2003), où les données sont décrites en définissant des éléments communs de données et un vocabulaire commun. Toutes les sources de données et outils doivent adhérer à ce modèle pour profiter de l'architecture. Les bénéfices de cette approche viennent du développement de standards permettant l'intégration de sources de données. En contre partie, les fournisseurs de données doivent fournir de gros efforts pour rendre leurs sources compatibles avec ce sstandards.
- Les **intégrations de vues** sont une approche d'intégration forte. Un environnement est construit permettant d'interroger les sources distribuées de manière transparente. Cette approche est rendue possible par l'utilisation d'un médiateur couplé à des adaptateurs faisant la correspondance entre le schéma d'intégration et les schémas des sources.
- Les **entrepôts de données** consistent à extraire, nettoyer et transformer les données issues de différentes sources, afin de les intégrer dans un système centralisé. Les données pourront ainsi être stockées et interrogées comme une source intégrée unique. Il s'agit d'une approche d'intégration forte.

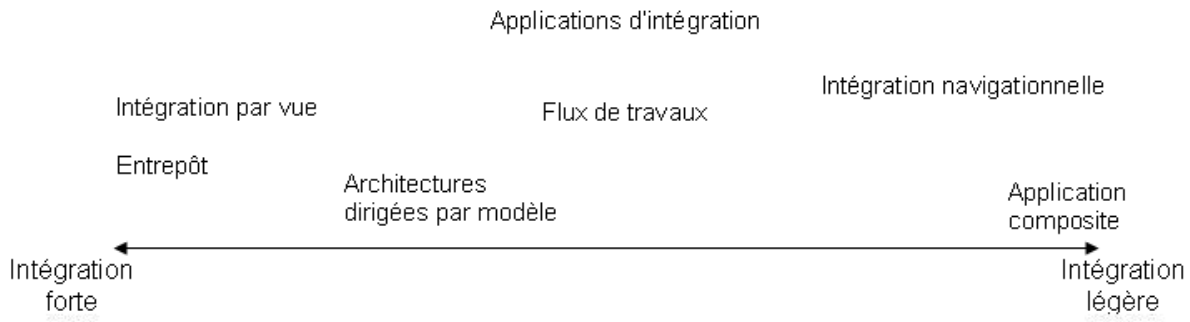


Figure 1 : Spectre des approches d'intégration de données (Carole Goble & Robert Stevens 2008)

1.2.2. L'entrepôt de données

Un entrepôt est défini comme étant « une collection de données orientées sujet, intégrées, non volatiles, historisées, organisées pour le support d'un processus d'aide à la décision » (Inmon 2005). Une intégration de type entrepôt consiste à matérialiser en local les données issues de sources de données multiples et d'exécuter toutes les requêtes sur les données contenues dans cet entrepôt local plutôt que dans les sources d'origine (Oladele 2008). Ces données pourront ainsi être interrogées à l'aide d'un unique langage de requête. Les approches de type entrepôt mettent l'accent sur la transformation de données, en opposition aux systèmes de médiation qui mettent l'accent sur la transformation de requête (X. Wang et al. 2009). En effet, les données à charger dans un entrepôt doivent dans un premier temps être converties pour être rendues compatibles avec le schéma global de l'entrepôt. Cette étape consiste à extraire la donnée de sa source d'origine puis à la transformer de manière à la rendre compatible avec le schéma global de l'entrepôt pour enfin la charger dans l'entrepôt. Cette étape est réalisée par un processus appelé ETL (Extract Transform and Load).

Dans le domaine biologiques il existe des entrepôts populaires développés pour des domaines spécifiques: GUS (Davidson et al. 2001), GIMS (M. Cornell et al. 2001), GEDAW (Guérin 2005) ou Atlas (S. P. Shah et al. 2005) pour le domaine génomique; Columba (Rother et al. 2004) pour le domaine protéique. Il existe également des boites à outils génériques et réutilisables offrant des fonctionnalités d'intégration : Biomart (J. Zhang et al. 2011), Biowarehouse (Lee et al. 2006) sont les plus répandues.

1.2.2.1.Les avantages

La centralisation des données permet d'éliminer les problèmes comme les temps de réponse trop long, les goulots d'étranglement du réseau ou encore l'indisponibilité temporaire de certaines sources. Cela permet d'effectuer des analyses complexes, trop lourdes pour être lancées à distance, mais également de tester de nouvelles techniques de fouilles de données. La copie locale des données permet également d'ajouter facilement des annotations ou de nettoyer et trier des informations menant ainsi à une intégration plus forte des instances (Cohen-Boulakia 2005).

1.2.2.2.Les inconvénients

La première difficulté de cette approche réside dans l'intégration de nouvelles sources. En effet les informations utilisées pour construire le schéma global de l'entrepôt, également appelé schéma intégrateur, sont dépendantes des sources à intégrer. La mise à jour de ce schéma peut devenir très compliquée, limitant ainsi l'ajout de nouvelles sources de données. Un autre inconvénient se situe au niveau de la synchronisation des données de l'entrepôt avec les données des sources. En effet, dans une approche de type entrepôt, il faut régulièrement vérifier les sources intégrées afin de trouver de nouvelles données ou des données mises à jour, et ainsi répercuter ces modifications sur la copie locale des données (Davidson et al. 1995). Pour cela il est nécessaire de ré-exécuter le processus ETL.

1.2.3. La médiation

La notion de médiation a été introduite par Wiederhold (G. Wiederhold & Genesereth 1997). Elle consiste à définir une interface entre un utilisateur qui pose une requête et l'ensemble des sources accessibles via le Web potentiellement pertinentes pour répondre (Gio Wiederhold 1992). Une intégration basée sur la médiation se concentre sur la transformation de requêtes. Cette approche consiste à construire un système d'intégration de données sans exporter les données de leur source d'origine, il donne ainsi l'impression à l'utilisateur d'interroger un système homogène et centralisé alors que les sources interrogées sont réparties, autonomes et hétérogènes. L'architecture globale d'un système de médiation est présenté dans la Figure 2, il est composé de 2 éléments principaux : le médiateur et les adaptateurs. Le médiateur est constitué d'un schéma global représentant le domaine d'application du système. Un utilisateur peut poser ses requêtes en utilisant les termes du schéma global. Le médiateur ne peut pas répondre directement à ces requêtes car il n'a pas

accès aux données à interroger mais à des vues des données contenues dans les sources à interroger. Les requêtes exprimées en termes du schéma global sont donc traitées par le médiateur qui va les réécrire en termes de vues des sources à interroger. Le médiateur récupèrera ensuite les résultats de chaque adaptateur et les combinera de manière à obtenir le résultat final. L'adaptateur est spécifique à chaque source de données, c'est lui qui définit les correspondances entre le schéma global du médiateur et le schéma local de la source de données.

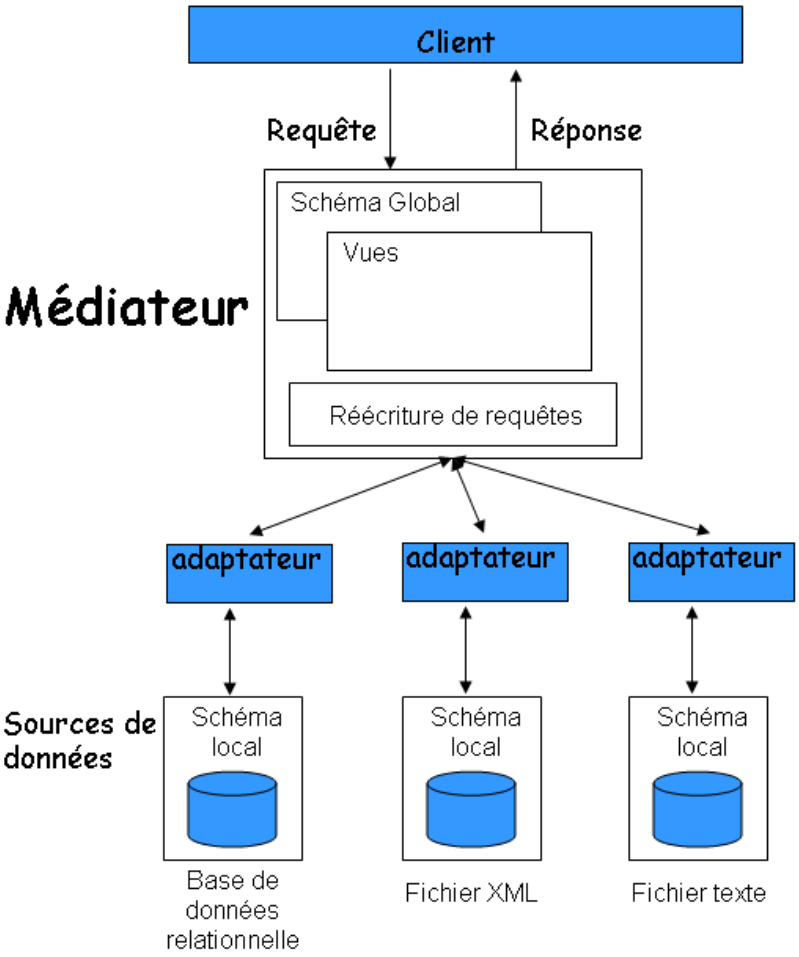


Figure 2 Architecture d'un médiateur

Les systèmes d'intégration de données de type médiation se distinguent par la manière dont est établie la correspondance entre les schémas des sources et le schéma global (A. Y. Levy 1999). Il s'agit d'une étape cruciale lors de la création du médiateur car elle influe sur la difficulté de reformulation de requêtes ainsi que sur la capacité d'ajouter ou de supprimer des nouvelles sources. Les deux approches principales utilisées pour établir les correspondances sont Global as View (GAV) (Halevy 2001) et Local as View (LAV) (Florescu et al. 1998). Il existe également des approches mixtes combinant les deux

précédentes comme GLAV (Global Local As View) (Lenzerini 2002) ou BGLAV (Both Global Local As View) (L. Xu & Embley 2007a).

L'approche Global as View (GAV) (Halevy 2001) consiste à définir le schéma global comme une vue des schémas locaux. Il s'agit de la première approche à avoir été utilisée pour l'intégration d'informations et provient du monde des bases de données fédérées. L'avantage de cette approche est la facilité de réécriture de requête. Cela est du au fait que l'approche GAV spécifie directement la correspondance entre une source à interroger et les éléments du schéma global (Lenzerini 2002). Par contre, cette approche rend l'ajout de nouvelles sources de données au système très complexe car les nouvelles relations risquent de modifier le schéma global.

L'approche Local as View (LAV) (Florescu et al. 1998) consiste à définir chaque schéma local comme une vue d'une portion du schéma global. Les avantages et inconvénients de cette approche sont inversés par rapport à l'approche GAV. Elle permet donc de rajouter facilement de nouvelles sources de données car cela n'a aucune incidence sur le schéma globale, mais par contre la réécriture de requête est beaucoup plus complexe.

1.2.3.1.Les adaptateurs

Pour qu'une source de données soit ajoutée au médiateur il faut dans un premier temps développer un adaptateur. La création d'un adaptateur est réalisée en deux étapes.

Le schéma des sources de données à intégrer n'est pas identique au schéma du médiateur, il faut donc déterminer dans un premier temps des règles de correspondances entre le schéma global et le schéma de la source à l'aide d'outils de mise en correspondance (mapping) entre schémas (L. Xu & Embley 2007b). Ce sont ces règles de correspondances qui seront utilisés par la suite pour transformer une requête posée dans le schéma global en requêtes compréhensibles par le schéma local.

La deuxième étape consiste à créer physiquement l'adaptateur permettant de renvoyer les données au médiateur et peut être réalisée automatiquement (Reynaud & Safar 2009).

La création d'adaptateur peut être complexe et consommatrice de temps. Elle doit être réalisée par le fournisseur de données qui doit souvent s'habituer à des outils ou des langages spécifiques au médiateur ce qui peut le rebuter et ainsi limiter le nombre de sources accessibles par le médiateur. Le développement de ces adaptateurs est plateforme dépendant, cela signifie qu'un adaptateur ne pourra être utilisé que pour une plateforme précise car les correspondances sont définies en fonction de son schéma global.

1.2.3.2.Les avantages

Les approches de médiation évitent à l'utilisateur de se soucier du choix des sources à interroger, lui permettant d'interroger de manière transparente plusieurs sources de données en ayant l'illusion d'un interroger une seule. Cela est d'autant plus important qu'un grand nombre de sources biologiques sont disponibles sur le Web (Michael Galperin & Cochrane 2011) et que l'utilisateur n'a pas le temps de les interroger toutes ou peut être pas connaissance de toutes les sources disponibles. L'ajout de nouvelles sources de données est plus simple que dans les approches de type entrepôt, et est encore plus simple dans le cas d'une approche LAV. Un autre avantage de cette approche est l'absence de gestion de mises à jour car les données sont interrogées directement dans leur source d'origine, ce qui est très intéressant dans notre contexte de sources de données biologiques évoluant rapidement.

1.2.3.3.Les inconvénients

Cette approche est dépendante de la disponibilité et de l'accessibilité des sources. En effet l'indisponibilité d'une source peut empêcher l'exécution de la requête. Par ailleurs, les temps d'exécution peuvent être longs en raison du caractère distribué des requêtes.

L'ajout de nouvelles sources de données consiste à créer un adaptateur faisant le pont entre le schéma global et le schéma local de la source à intégrer. Cette étape peut être consommatrice de temps et représente le goulot d'étranglement à l'ajout de nouvelles sources de données dans un médiateur.

1.2.3.4.Les solutions existantes dans le domaine bioinformatique

Plusieurs approches de médiation ont été mises en place en biologie : K2/BioKleisli (Davidson et al. 2001), TAMBIS (R. D. Stevens et al. 2000), DiscoveryLink (L. M. Haas et al. 2001) ou GCP Pantheon (Richard Bruskiewich, Martin Senger, G. Davenport, Ruiz, Rouard, Hazekamp, et al. 2008). Nous n'allons pas présenter les différences entre ces différents systèmes, des états de l'art très bien réalisés étant disponibles (Oladele 2008; Larmande 2007).

1.3. Conclusion

Il n'existe pas de solution idéale pour l'intégration de données. La solution la plus appropriée dépend des besoins à couvrir par cette intégration. Les solutions d'intégration proposant le couplage le plus fort sont les approches d'entrepôt de données et de médiation.

Un entrepôt de données est intéressant quand les sources à intégrer sont clairement identifiées et qu'aucune source supplémentaire ne sera rajoutée après la création du schéma global. Il permet une interrogation rapide car toutes les sources sont centralisées. Cependant, les entrepôts posent des problèmes de gestion de mise à jour des données notamment dans le cas de sources de données fortement évolutives comme c'est le cas dans le domaine de la biologie.

Les approches de médiation permettent de rajouter plus facilement de nouvelles sources de données, notamment dans le cas d'une approche LAV. L'interrogation des sources dans leur localisation d'origine évite les problèmes de gestion de mise à jour des données mais en contrepartie diminue la vitesse d'interrogation.

L'ajout de nouvelles sources de données dans une approche de médiation nécessite la création d'un adaptateur situé à l'interface entre le schéma global et le schéma local et permettant de transformer une requête exprimée dans les termes du schéma global en une requête compréhensible par la source de données. Le développement de ces adaptateurs doit être réalisé par le fournisseur de données. Il peut être consommateur de temps ce qui peut rebuter le fournisseur, et au final, limiter le nombre de sources disponibles dans le système de médiation.

La création d'un schéma global spécifique au système d'intégration limite la réutilisation de ces adaptateurs. Les approches du Web Sémantique pourraient conduire à la création d'adaptateurs réutilisables. En plus de problèmes de sémantiques des adaptateurs, leur structure même peut ne pas être compatible pour d'autres utilisations. Les Services Web, sont des programmes informatiques permettant la communication et l'échange de données entre applications et systèmes hétérogènes indépendants de tout langage de programmation et de toute plateforme d'exécution. La combinaison des approches de Services Web et du Web Sémantique est intéressante pour dépasser les défis actuels d'intégration des bases de données biologiques tels que la syntaxe, la sémantique, l'interopérabilité et le dynamisme (Oladele 2008).

2. Le Web Sémantique

2.1. Introduction

Le point commun de l'intégration de données et des technologies du Web Sémantique est de dépasser l'hétérogénéité sémantique de sources de données interconnectées. L'utilisation du Web Sémantique facilite la représentation de la sémantique des données et peut ainsi être utilisé pour faciliter l'interopérabilité ou l'intégration de données.

Il est courant de parler de Web Sémantique lorsque l'on souhaite parler des technologies et langages standards qui le composent. Ces standards peuvent être largement utilisés en biologie pour favoriser l'intégration de sources de données (Antezana, Kuiper, et al. 2009).

Dans une première partie nous allons présenter brièvement l'architecture globale du Web Sémantique. Nous détaillerons ensuite le langage RDF, standard de la représentation de données. Dans une quatrième partie nous introduirons la notion de web de données (*linked data*). La partie suivante traitera des ontologies ainsi que du langage standard OWL, puis nous introduirons le langage de requête standard SPARQL.

2.2. L'architecture du Web Sémantique

En tant que technologie, le Web Sémantique peut être résumé comme « la représentation de connaissance qui rencontre le Web » (Grosof et al. 2003). Le but est de créer des langages, permettant le traitement automatique et la composition d'informations sur le Web. Le W3C (World Wide Web Consortium) est un organisme de standardisation chargé de promouvoir la compatibilité des technologies du Web, c'est lui qui recommande les standards qui seront utilisés dans les différentes couches de l'architecture du Web Sémantique. Cette architecture est composée de plusieurs briques technologiques. La Figure 3 est une représentation sous forme de pièce montée (layer cake) de cette architecture (W3C s. d.).

Nous allons présenter succinctement chaque brique de cette architecture. OWL, RDF et SPARQL seront présentés en détail plus tard.

- **URI** (Uniform Resource Identifier) (Berners-Lee 1994) : permet d'identifier la localisation d'une donnée et la méthode pour y accéder. Dans le Web Sémantique

l'URI est utilisée pour identifier les données et permettre ainsi à un agent d'aller chercher la donnée là où elle est stockée.

- **XML** (eXtensible Markup Language) (Bray et al. 2008) : est un langage informatique de balises permettant d'encoder un document et pouvant être exploité par une machine. Les documents XML sont composés d'unités de stockage appelées des entités, elles mêmes contenant des données pouvant ou non être parsées.
- **RDF** (Resource Description Framework) (RDF Working Group 2004) : est le modèle standard pour la description de données sur le Web. Il possède des caractéristiques facilitant la fusion de données même si les schémas sous jacents sont différents. Il supporte l'évolution des schémas au cours du temps sans nécessiter de modification de la part des consommateurs de données.
- **RDFS** (RDF Schema) (Brickley & Guha 2004) : est un langage de déclarations et de descriptions légères des types de ressources (appelés Classes) et de leurs relations (appelées Propriétés). RDFS permet de nommer et définir un vocabulaire utilisé dans des graphes d'annotations. Il permet aussi des inférences élémentaires notamment en exploitant les liens hiérarchiques entre classes ou les liens hiérarchiques entre propriétés. Il offre toute l'expressivité nécessaire à l'élaboration de vocabulaires. Parmi ces vocabulaires, on peut notamment citer Dublin Core (Hakala 2000) qui permet de décrire des ressources numériques ou physiques et d'établir des relations avec d'autres ressources ; FOAF (Friend Of A Friend) (Brickley & L. Miller 2010) permettant de décrire des personnes ainsi que les relations entre elles ou encore SKOS (Simple Knowledge Organisation System) (Summers & Phipps 2008) permettant de définir des systèmes d'organisation de la connaissance.
- **OWL** (Web Ontology Language) (McGuinness & Harmelen 2004) : est un langage apportant une extension de l'expressivité logique de RDFS. Cela signifie que le langage OWL permet d'exprimer plus d'idées que le langage RDFS. Il est destiné à être utilisé quand l'information contenue dans un document doit être traitée par une application plutôt que par un humain. Il peut être utilisé pour représenter explicitement le sens des termes d'un vocabulaire et les relations entre ces termes.
- **SPARQL** (Prud'hommeau & Seaborne 2008) : est le langage de requête pour RDF, il s'agit du maillon essentiel permettant de manipuler les données.
- **RIF** (Rule Interchange Format) (Boley & Kifer 2010) : est le dernier standard de cette architecture à avoir été adopté par le W3C. Il permet de supporter des règles, qui

autorisent la description de relations ne pouvant pas être décrite directement à l'aide de OWL.

Les briques Confiance (Trust), Preuve (Proof) et Cryptage (Crypto) contiennent des idées qui devront être implémentées pour obtenir un Web Sémantique compatible avec la vision initiale de Tim Berners-Lee. La partie Cryptage correpsond à une signature digitale permettant de vérifier que les données proviennent d'une source de confiance. La partie Preuve doit permettre de s'appuyer sur une logique formelle pour détecter de nouvelles informations. La partie Confiance s'appuiera sur les parties Preuve et Cryptage pour déterminer la confiance des données récupérées. A l'heure actuelle aucun standard permettant de répondre à ces besoins n'a été recommandé par le W3C.

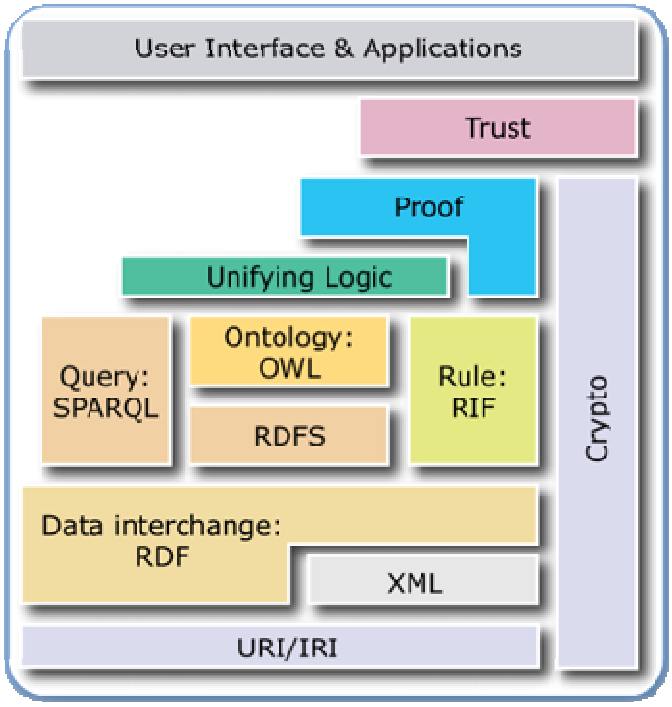


Figure 3 : Architecture générale du Web Sémantique (W3C s. d.)

2.3. Le langage RDF

RDF est le langage standard du Web Sémantique permettant de représenter des métadonnées comme par exemple le titre, l'auteur ou la date de modification d'une page Web, les droits d'auteur, les informations de licence d'un document Web, les critères de disponibilité ou les paramètres de confidentialité d'une ressource partagée. Il utilise des idées bien établies issues de différentes communautés de représentation de données et de

connaissances, avec des approches liées aux graphes conceptuels (J. F. Sowa 1984), aux représentations de connaissances basées sur la logique (J. E. Sowa & Shapiro 2006) et aux bases de données relationnelles (Gray 1984). RDF est basé sur XML qui fournit une structure syntaxique pour représenter des documents et d'autres informations mais qui ne fournit pas d'informations sur la sémantique des données. RDF possède un modèle de graphe simple et une sémantique formelle (P. Hayes 2004) avec une notion d'implication définie rigoureusement, fournissant une base pour des déductions sur les données RDF (Klyne & Carroll 2004). Cela rend possible le traitement automatique d'une ressource Web à l'aide d'agents logiciels, faisant ainsi passer le Web de support d'informations lisibles par les humains à un réseau de processus coopératifs. Le standard RDF permet également aux développeurs d'applications de profiter de la présence de parseurs RDF génériques. La possibilité d'échanger de l'information entre différentes applications implique qu'une information créée spécifiquement pour une application peut être utilisée dans toute autre application s'appuyant sur du RDF, ce qui favorise le partage d'information.

2.3.1. L'identifiant unique

RDF est basé sur l'idée de pouvoir tout identifier en utilisant une URI. L'utilisation d'URIs permet de relier des fragments d'informations dispersés sur le Web. Dans le langage RDF, une URI peut être utilisée pour renvoyer à toute chose identifiable, y compris celles qui ne peuvent pas être récupérées directement sur le net, comme par exemple une personne, une plante ou un gène. Les URIs sont utilisées pour identifier précisément une ressource, qu'il s'agisse d'une classe ou d'une propriété. Dans l'exemple ci-dessous, la première ligne permet d'identifier précisément la classe « Person » issue du vocabulaire FOAF. La deuxième ligne de cet exemple définie la propriété « type », issue de RDF. Chacune de ces ressources est une URI de référence, composée d'une URI identifiant le contexte, puis d'un nom, présenté ici en gras, définissant le terme utilisé pour le contexte choisi.

Exemple :http://xmlns.com/foaf/0.1/**Person**
http://www.w3.org/1999/02/22-rdf-syntax-ns#**type**

2.3.2. Le triplet RDF

Les triplets sont les éléments de base du langage RDF permettant de décrire et de connecter des ressources, c'est-à-dire des objets anonymes ou identifiés par une URI. Chaque triplet est composé d'un sujet, d'un prédicat et d'un objet. Le prédicat, également appelé

propriété, permet de définir une relation entre le sujet et l'objet. Un triplet peut être visualisé comme un ensemble de 2 nœuds reliés par un arc orienté et labélisé, comme c'est le cas dans la Figure 4.

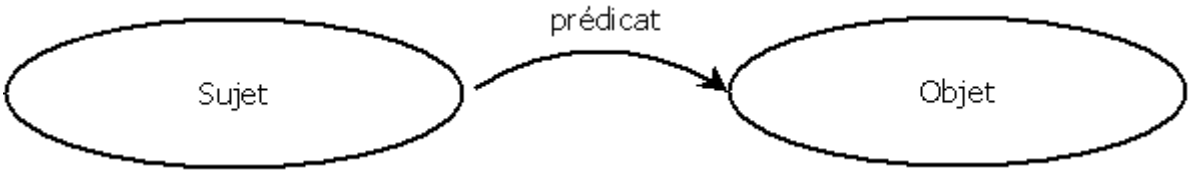


Figure 4 : Représentation graphique d'un triplet

2.3.3. Le graphe RDF

Un graphe RDF est un ensemble de triplets. Chaque nœud de ce graphe est le sujet ou l'objet d'un triplet. Le graphe RDF de la Figure 5 indique qu'une ressource dont le titre est BioSemantic a pour créateur Julien Wollbrett qui est de type Person et qui possède l'adresse mail julien.wollbrett@cirad.fr. Les triplets correspondants à ce graphe sont présentés dans la Figure 6. Les nœuds représentés en vert correspondent à des ressources et les nœuds orange correspondent à des littéraux, c'est-à-dire des nœuds ayant une valeur (e.g chaîne de caractères, chiffre). Un littéral ne peut pas être le sujet d'un triplet, cela signifie qu'il correspondra forcément à une *feuille* du graphe RDF.

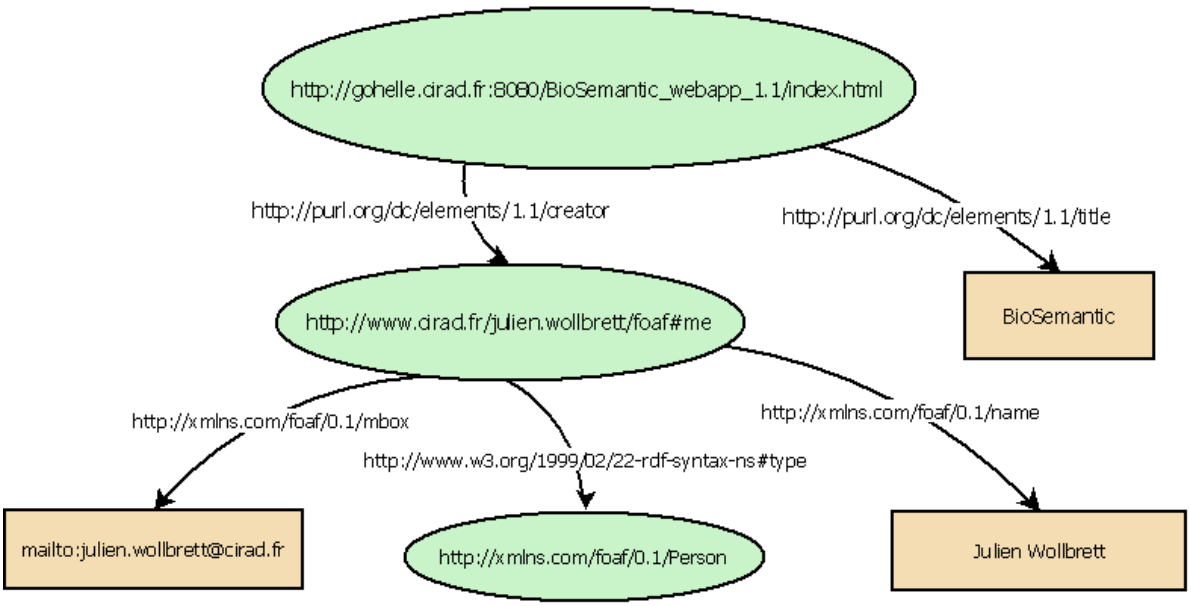


Figure 5 : Un exemple de graphe RDF

Sujet PrédicatObjet
<http://gohelle.cirad.fr:8080/BioSemantic_webapp_1.1/index.html> dc:creator <http://www.cirad.fr/jwollbrett/foaf#me>
<http://gohelle.cirad.fr:8080/BioSemantic_webapp_1.1/index.html> dc:title "BioSemantic"
<http://www.cirad.fr/jwollbrett/foaf#me> rdf:type foaf:Person
<http://www.cirad.fr/jwollbrett/foaf#me> foaf:name "Julien Wollbrett"
<http://www.cirad.fr/jwollbrett/foaf#me> foaf:mbox "mailto:julien.wollbrett@cirad.fr"

Figure 6 : Triplets correspondants au graphe de la Figure 5

2.3.4. La sérialisation

RDF est un modèle de données qui est indépendant de tout format de stockage. Des sérialisations ont donc été développées permettant de transformer le modèle RDF dans un format de fichier pouvant être stocké. Il existe 3 syntaxes permettant de sérialiser du RDF : RDF/XML (Dave Beckett & McBride 2004), N3 (Berners-Lee & Connolly 2011) et Turtle (David Beckett & Berners-Lee 2011). La syntaxe RDF/XML, est basée sur la syntaxe XML. Elle est recommandée pour le RDF. La Figure 7 est la représentation de la Figure 5 en utilisant cette syntaxe.

```
<rdf:RDF xmlns:dc="http://purl.org/dc/elements/1.1/"
   xmlns:foaf="http://xmlns.com/foaf/0.1/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
      <rdf:Description rdf:about="http://www-sop.inria.fr/edelweiss/fabien/docs/w3c/rdfsource/rdfsource.html">
      <dc:title>RDF Source</dc:title>
      <dc:creator>
         <foaf:Person rdf:about="http://ns.inria.fr/fabien.gandon/foaf#me">
            <foaf:name>Fabien Gandon</foaf:name>
            <foaf:mbox rdf:resource="mailto:fgandon@inria.fr"/>
         </foaf:Person>
      </dc:creator>
   </rdf:Description>
</rdf:RDF>
```

Figure 7 : syntaxe RDF/XML

La syntaxe N3 est une sérialisation de RDF proposée par Tim Berners-Lee. Elle est plus facilement lisible par des humains et est plus compacte que la sérialisation RDF/XML. Sa syntaxe permet de repérer facilement les triplets et est donc plus intuitive pour imaginer le graphe correspondant. La Figure 8 est la représentation du graphe de la Figure 5 en utilisant la syntaxe N3.

```
@prefix dc : < http://purl.org/dc/elements/1.1>
@prefix foaf : <http://xmlns.com/foaf/0.1/>
@prefix rdf : <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
<http://gohelle.cirad.fr:8080/BioSemantic_webapp_1.1/index.html>
dc:creator  <http://www.cirad.fr/jwollbrett/foaf#me>;
dc:title  "BioSemantic" .
<http://www.cirad.fr/jwollbrett/foaf#me>
rdf:type     foaf:Person;
foaf:name    "Julien Wollbrett";
foaf:mbox    "mailto:julien.wollbrett@cirad.fr" .
```

Figure 8 : syntaxe N3

Dans la suite de ce mémoire, toutes les illustrations montrant du RDF seront sérialisées en N3.

Le point commun des syntaxes RDF/XML et N3 est la description de préfixes au début du fichier. Dans la Figure 9 "dc" est défini comme étant le préfixe correspondant à l'URI http://purl.org/dc/elements/1.1/title, elle même étant l'URI correspondant au vocabulaire RDF appelé dublin core. Dans la suite du fichier, toutes les ressources issues de ce vocabulaire seront précédées de ce préfixe, évitant ainsi de répéter continuellement l'URI.

```
@prefix dc: http://purl.org/dc/elements/1.1/
```

Figure 9 : Description d'un préfixe

2.3.5. L'annuaire RDF

Un annuaire RDF que l'on appelle aussi triple store, est un gestionnaire de triplets RDF. Il s'agit d'une base de données permettant de stocker et renvoyer des données au format RDF. Les données peuvent être stockées de différentes façons selon les annuaires RDF. Certains stockent les données dans une base de données relationnelle, d'autres les stockent de façon native, c'est-à-dire directement sous forme de triplets, d'autres utilisent des fichiers XML ou implémentent leur propre stockage et format d'indexation. Certains annuaires RDF permettent également de manipuler et modifier les triplets stockés, c'est notamment le cas de Sesame (Broekstra et al. 2001), TDB[5] et 4Store (Garlik 2009).

Il existe plusieurs Benchmark permettant de déterminer les performances d'annuaires RDF, les plus utilisés sont LUBM (Y Guo et al. 2005) et BSBM (Chris Bizer & Schultz 2011a). LUBM a été développé pour faciliter l'évaluation d'annuaires du Web Sémantique

[5] http://openjena.org/TDB/

d'une manière standard et systématique. BSBM permet de comparer les performances de stockage de fichiers RDF ainsi que les performances de requête de bases de données relationnelles interrogées via le langage de requête SPARQL. Plusieurs versions du Benchmark BSBM sont disponibles (Chris Bizer & Andreas Schultz 2009; Chris Bizer & Schultz 2011b), mettant en avant les très bonnes performances de l'annuaire RDF Sesame lorsque peu de triplets sont stockés (1 million de triplets) ainsi que la domination de l'annuaire Virtuoso pour les gros jeux de données (100 ou 200 millions de triplets)

2.4. Le langage de requête SPARQL

SPARQL (SPARQL Protocol And RDF Query Language) (Prud'hommeaux & Seaborne 2008) est un langage de requête RDF. Il a été proposé en 2004 (Prud'hommeaux & Seaborne 2004) par Eric Prud'hommeaux et Andy Seaborn puis recommandé par le W3C en 2008. Il est actuellement considéré comme une technologie clé du Web Sémantique. Il permet de requêter des graphes RDF en posant des requêtes sous forme de triplets. En plus d'offrir un langage de requêtes, SPARQL fournit également un langage de résultats (Dave Beckett & Broekstra 2008) permettant de représenter les réponses d'une requête ainsi qu'un protocole (Clark et al. 2008) permettant de soumettre une requête à un serveur distant. La Figure 10 est un exemple de requête SPARQL simple posée sur le graphe de la Figure 5. Dans une requête SPARQL les variables sont précédées d'un "?". La requête peut être divisée en trois parties. L'identification des préfixes, permettant de ne pas répéter les URI dans toute la requête, la clause SELECT qui identifie les variables à renvoyer dans la réponse, et la clause WHERE qui défini les conditions à respecter, écrites sous forme de triplet. Dans notre exemple, nous recherchons une variable ?title qui correspond à la propriété "dc:title" de la ressource `<http://gohelle.cirad.fr:8080/BioSemantic_webapp_1.1/index.html>`. La réponse à cette requête pour le graphe RDF sélectionné est "BioSemantic".

```
@prefix dc:   <http://purl.org/dc/elements/1.1/#>.
SELECT ?title WHERE {
<http://gohelle.cirad.fr:8080/BioSemantic_webapp_1.1/index.html>  dc:title  ?title.
}
```

Figure 10 : Exemple d'une requête SPARQL Simple

Les requêtes SPARQL sont posées principalement sur des points d'entrée (endpoint) SPARQL. Un point d'entrée SPARQL est un service SPARQL conforme au protocole défini dans (Clark et al. 2008). Il permet à un utilisateur, humain ou non, de requêter une base de connaissance à l'aide du langage SPARQL. Les résultats de ces requêtes sont renvoyés dans des formats utilisables par des machines. La plupart des annuaires RDF proposent un point d'entrée SPARQL.

2.5. Le Web des Données (Linked Data)

Le Web Sémantique ne consiste pas uniquement à mettre des données sur le Web. C'est en faisant des liens qu'une personne ou une machine peut explorer le Web des Données. Avec des données reliées, quand vous avez une portion de ces données, il est possible de trouver automatiquement d'autres données associées sémantiquement. De la même manière que le Web et ses liens hypertextes, le Web des Données est construit à partir de documents disponibles sur le Web. La différence vient du fait que les liens hypertextes représentent des relations entre des documents écris en HTML alors que pour le Web des Données les liens sont fait entre des « choses » arbitraires décrites en RDF (Berners-Lee 2006).

Tim Berners-Lee a décrit 4 règles à respecter pour rendre des données disponibles sur le Web des Données (Berners-Lee 2006). Ces 4 règles sont :

- utiliser les URIs pour nommer les « choses »,
- utiliser des URIs du protocole HTTP (R. Fielding et al. 1999) pour permettre d'accéder à des définitions de ces « choses »
- fournir à travers l'URI des renseignements lisibles par les humains et par les machines,
- inclure des liens vers d'autres URI de manière à découvrir plus d'informations à travers le Web.

Un tutoriel très complet expliquant les marches à suivre pour publier ces données sur le Web des Données est également disponible (Chris Bizer et al. 2007). Le nombre de sources disponibles via le Web des Données est en constante évolution. La Figure 11 (Cyganiak & Jentzsch 2011) est un diagramme dans lequel chaque nœud correspond à une source de données disponible sur le Web des Données. On peut y distinguer 7 domaines différents, représentés chacun par couleur, parmi lesquels se trouve la biologie, représenté par des nœuds de couleur parme.

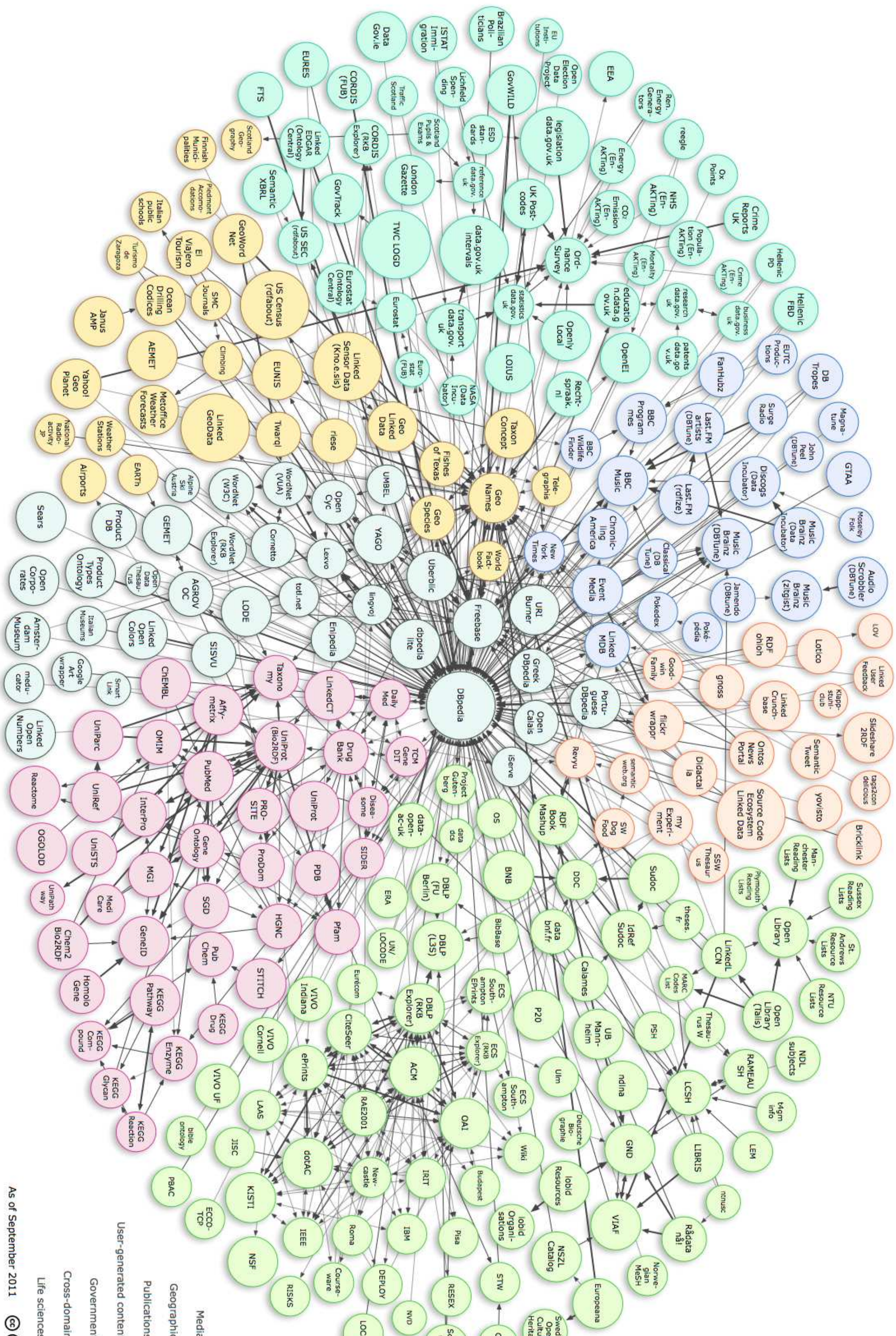


Figure 11 : Diagramme représentant les sources disponibles via le Web des Données
(Cyganiak & Jentzsch 2011)

2.6. Les ontologies

Le mot ontologie vient du domaine de la philosophie où il est utilisé pour définir l'étude de l'être, de ses modalités et de ses propriétés, c'est-à-dire l'étude de la nature de tout ce qui existe. Le terme « ontologie » a été employé dans des domaines très différents comme l'intelligence artificielle ou encore la linguistique. Dans la communauté informatique, Gruber (Gruber 1993) définit le premier une ontologie comme étant « la spécification d'une conceptualisation ». Une ontologie correspond à la description des concepts et des relations sémantiques ou d'inclusions entre ces concepts pour un domaine de connaissances donné.

Une ontologie peut être définie comme le modèle d'une base de connaissance. L'expressivité des langages ontologiques est plus fermée à la logique de premier ordre (FOL) (Hodges 2001) que le sont les langages utilisés pour modéliser des bases de données (Konstantinou et al. 2008). C'est pour cette raison que les ontologies sont considérées comme étant au niveau sémantique alors que les schémas de bases de données sont situés au niveau logique ou physique

2.6.1. Les logiques de description

Les logiques de description (LD) constituent une famille de langages de représentation de connaissances fondée sur un formalisme logique. Elles sont issues d'un courant de recherche initié par le système KL-ONE (Schmolze et al. 1985) permettant de surmonter les ambiguïtés sémantiques que présentaient les approches de l'époque basées sur les réseaux sémantiques (Woods 1975) ou encore les langages de frame (Fikes & Kehler 1985).

Une base de connaissances en logique de description est composée de deux éléments différents : la **TBox** et la **ABox**.

La TBox constitue une terminologie, c'est-à-dire le vocabulaire d'un domaine d'application. Ce vocabulaire est constitué de concepts qui correspondent à un ensemble d'individus et peuvent être comparés aux prédicats unaires des logiques de prédicats, et de rôles qui représentent des relations binaires entre les individus (Baader et al. 2003).

La ABox constitue la composante assertionnelle, elle représente un état particulier du domaine de la TBox.

2.6.2. Les langages d'ontologie

Il existe différents langages formels utilisés pour construire des ontologies. Ces langages permettent d'encoder la connaissance d'un domaine spécifique et incluent souvent des règles de raisonnement permettant de traiter cette connaissance.

2.6.2.1.Le langage OWL

Le langage OWL (Web Ontology Language) (Sean Bechhofer et al. 2004) est un langage sémantique permettant de publier et de partager des ontologies sur le Web. C'est le langage recommandé par le W3C s'appuyant sur les logiques de description. Il a été développé comme une extension des vocabulaires RDF et RDFS. Une évolution du langage OWL, appelée OWL 2, a été recommandée par le W3C en 2009 (Motik et al. 2009) apportant de nouvelles fonctionnalités. Beaucoup de ces nouvelles caractéristiques sont syntaxiques (e.g union disjointe de classe) mais certaines offrent plus d'expressivité (e.g des types de données plus riches, des propriétés de réflexivité et d'asymétrie). OWL 2 possède 5 profils différents (OWL Lite, OWL DL, OWL 2 QL, OWL 2 EL et OWL 2 RL), chacun possédant sa propre expressivité. Le choix d'un profil dépend du niveau d'expressivité souhaité.

2.6.2.2.Le langage OBO

OBO (Open Biomedical Ontologies) est un langage utilisé initialement par le Consortium Gene Ontology pour la création de son ontologie (Michael Ashburner et al. 2000). Il s'agit d'un langage couramment utilisé pour créer des ontologies dans le domaine biologique, et dont la sémantique est définie de façon informelle (Golbreich et al. 2008), même si des conventions typographiques, syntaxiques et sémantiques ont été proposées (Schober et al. 2009). Les ontologies OBO sont stockées dans un format de fichier à plat appelé format OBO (Chris Mungall & Ireland 2010). Ce format a été créé dans l'optique d'être facilement lisible par des humains, facile à parser, extensible et contenant un minimum de redondances. Le langage OBO, ainsi que les outils l'utilisant, ont été développés indépendamment des standards du Web Sémantique. Il serait intéressant de rendre les formats OBO et OWL compatibles permettant ainsi à la communauté OBO de réutiliser les outils OWL mais également pour créer un plus gros pool d'ontologies accessible à la communauté Web Sémantique (Golbreich et al. 2008). C'est dans ce but que des correspondances entre le format OWL et le format OBO ont été décrites explicitement, permettant de transformer une ontologie OBO en une ontologie OWL (Golbreich & Horrocks 2007).

Les principales ontologies OBO utilisées dans le domaine de la diversité génétique des plantes sont les suivantes :

- *Gene Ontology* (GO) est la première bioontologie développée dans le format OBO. Il s'agit d'une initiative bioinformatique majeure dont le but est de standardiser la représentation de gènes et de produits de gènes à travers les bases de données biologiques quelle que soit l'espèce concernée. Ce projet fournit un vocabulaire contrôlé permettant de décrire les caractéristiques de produits de gènes et d'annoter les données de produits de gènes. GO est plus précisément composée de trois ontologies différentes couvrant trois domaines de connaissance différents: les processus biologiques, les fonctions moléculaires, et les localisations ou composants cellulaires.

- *Sequence Ontology* (SO) (K Eilbeck et al. 2005) offre un ensemble de termes et de relations permettant de décrire les caractéristiques et les propriétés de séquences biologiques.

- *Plant Ontology* (PO) (Jaiswal et al. 2005) est développée par le *Plant Ontology Consortium*. Elle décrit l'anatomie des plantes, leur structure morphologique, leur croissance et leurs différentes étapes de développement. Le but du consortium est d'établir un cadre sémantique pour la création de requêtes d'expression de gènes et de données phénotypiques chez les plantes.

- *Trait Ontology* (TO) (Jaiswal et al. 2005) est aussi développée par le *Plant Ontology Consortium*. Elle permet la description des caractères phénotypiques d'une plante en croissance ou mature. Ces caractères regroupent notamment les notions de vigueur, d'anatomie et de morphologie, de biochimie, de fertilité, de développement ou encore de taille.

- *Crop Ontology* (CO) (Shrestha et al. 2010) est développée dans le cadre du projet GCP. Elle permet la description des caractères phénotypiques de plantes cultivés comme le riz, le maïs, le blé, la pomme de terre, le sorgho, le poids chiche et la banane. Elle inclut des informations liées aux données de ressources génétiques.

2.6.3. Les logiciels d'édition d'ontologies

La création d'ontologie est une tâche lourde pouvant conduire à de nombreuses erreurs (N. F Noy et al. 2001). Une raison de cette difficulté est que les ontologies sont des modèles formels du domaine de connaissance. Comme la connaissance humaine est souvent implicite ou difficile à décrire, il est difficile de la structurer formellement. Bien qu'il existe des règles à suivre permettant de faciliter la création d'une nouvelle ontologie (Gruber 1993; Paslaru et al. 2006; Bergman 2010; OBO Foundry 2008), il est important de pouvoir créer des

ontologies en s'affranchissant des contraintes dues à son format grâce notamment à l'utilisation d'interfaces utilisateur conviviales.

Protégé (Knublauch et al. 2004) est un logiciel de création et de visualisation d'ontologies OWL ou à base de Frame. Il s'agit du logiciel d'édition d'ontologies le plus utilisé. Il permet notamment de créer des ontologies OWL 2, d'utiliser un raisonneur permettant de tester la satisfiabilité des définitions de classes, c'est-à-dire de vérifier si les définitions de ces classes sont correctes en terme de logique de description. Protégé permet également de détecter des relations restées implicites dans l'ontologie, ou encore de réaliser des requêtes SPARQL. Il s'agit d'un outil open source. Une librairie ainsi que des tutoriaux bien documentés permettent aux utilisateurs de créer facilement leur propre plugin. De nombreux plugins on été développés parmi lesquels nous pouvons noter

oboConverter : qui permet de transformer une ontologie au format OBO vers une ontologie au format OWL,

Prompt : qui permet de vérifier les points communs entre 2 ontologies. Il est capable de fusionner 2 ontologies ou encore de vérifier les différences entre 2 versions d'une même ontologie,

DataMaster : qui permet de se connecter à n'importe quelle base de données relationnelle pour importer son schéma.

OBO-Edit (Day-Richter et al. 2007) est un logiciel de création et de visualisation d'ontologies optimisé pour le format OBO. Il permet de visualiser une ontologie sous forme graphe acyclique orienté et contient un raisonneur.

Jena[6] est une structure permettant de créer des applications Web Sémantique. Il fournit un environnement de programmation pour les standards RDF, RDFS, OWL, SPARQL et inclus un moteur d'inférence. Actuellement Jena n'est pas compatible avec le langage OWL 2.

OWL API (Horridge & Sean Bechhofer 2009) est une API Java fournissant un environnement de programmation de référence pour créer et manipuler des ontologies dans les langages OWL et OWL 2. Actuellement cette API n'est pas compatible avec SPARQL.

2.6.4. Les annuaires d'ontologies

Les ontologies et les vocabulaires contrôlés sont des ressources clés pour la création de métadonnées sur le Web Sémantique. Des annuaires regroupant les ontologies d'un

[6] http://jena.sourceforge.net/documentation.html

domaine d'application spécifiques permettent de faciliter leur utilisation et leur détection. Il y a un besoin véritable de créer différents types d'annuaires ontologiques, chacun se focalisant sur les besoins spécifiques de ces utilisateurs, sur différentes ontologies et différents besoins organisationnels qui ne peuvent pas être traitées par un unique annuaire (Viljanen et al. 2011). Dans le domaine biologique, il existe trois annuaires ontologiques importants :

- BioPortal est un annuaire d'ontologies biomédicales OWL développé par le NCBO (National Center of Biomedical Ontology), qui a été utilisé pour publier plus de 200 bioontologies (P. Whetzel et al. 2009). Il permet notamment de rechercher des concepts ou des ontologies, de les visualiser ou encore de les télécharger. Il permet également de commenter les concepts existants, ou encore de créer des correspondances entre des concepts, ou des concepts avec des sources de données, facilitant la détection de données rattachées à un concept.
- La communauté OBO Foundry (B. Smith et al. 2007) fournit une liste de toutes les bioontologies disponibles, précisant leur domaine, leur préfixe et la date de la dernière modification. Les ontologies référencées peuvent être au format OBO ou OWL. Par contre il n'y a pas d'outil de visualisation intégré.
- Ontology lookup Service (Côté et al. 2006) a été créé pour intégrer les ontologies biomédicales disponibles dans une unique base de données. Cette base de données permet d'obtenir des informations sur des concepts ontologiques ou de visualiser une ontologie complète. Il permet d'enregistrer des ontologies au format OBO. Il s'agit d'un projet open source, il est donc possible de réutiliser le code pour créer facilement son propre annuaire d'ontologies OBO.

2.7. La biologie et le Web Sémantique

2.7.1. Le Web des Données

Plus d'un dixième des sources disponibles sur le Web des Données sont issues du domaine de la biologie. La Figure 12 est un zoom de la Figure 11 centré sur les sources de données biologiques, et permettant de visualiser clairement le nom des sources biologiques disponibles sur le Web des Données. Les flèches permettent de visualiser les sources reliées entre elles. Les sources les plus utilisées pour relier les données biologiques entre elles sont le moteur de recherche de données bibliographiques PubMed, la base de données protéiques UniProt, la base de données regroupant les médicaments ainsi que leurs cibles Drug Bank, et la base de données regroupant les numéros d'accession des gènes GeneId.

La première source de données biologique à avoir été transformée en RDF et rendu disponible sur le Web des Données fut la base de connaissance UniProt (Redaschi & U. Consortium 2009). La transformation en RDF a été documentée[7], détaillant précisément le passage de la version UniProt au format texte vers un document RDF.

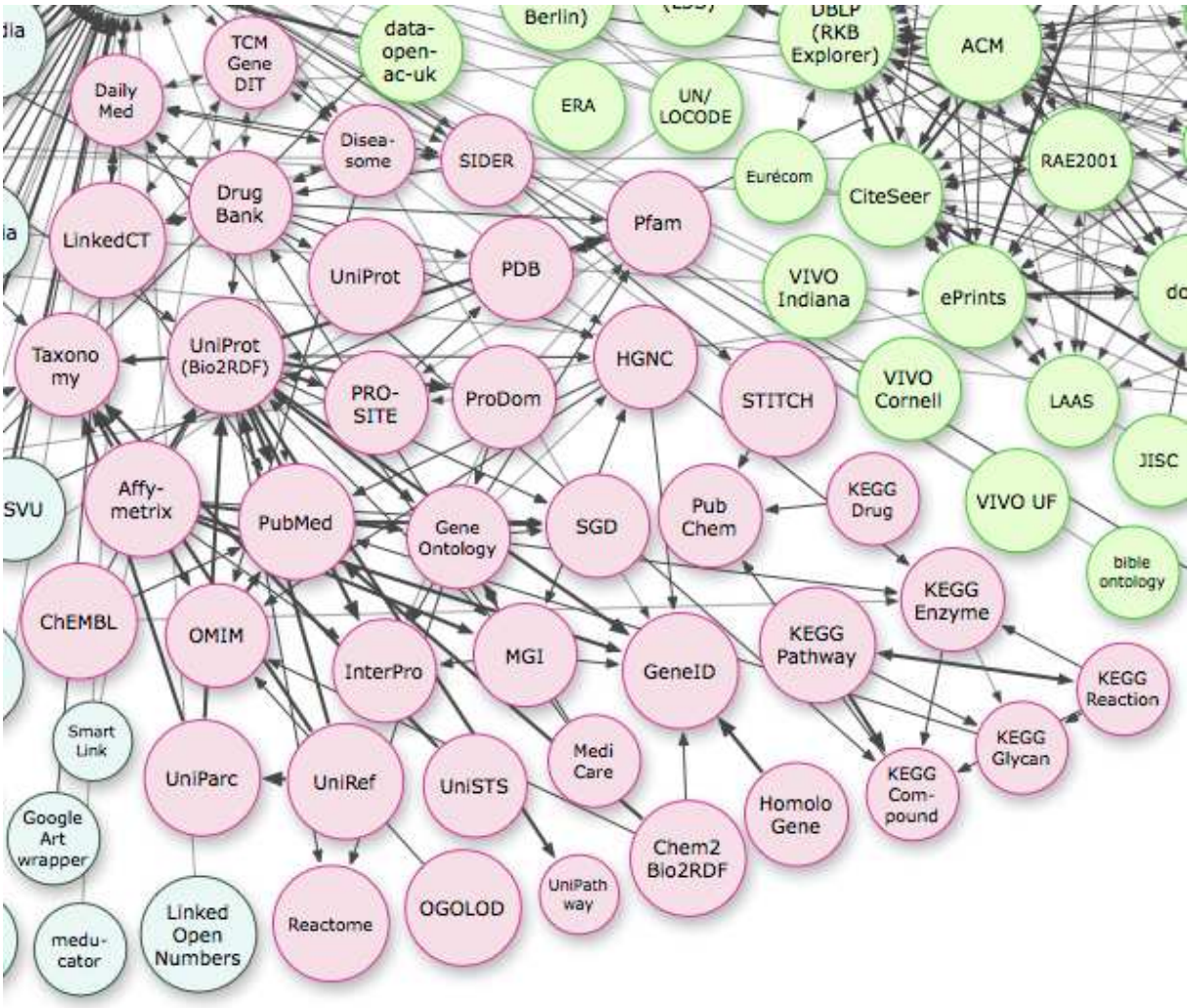


Figure 12 : Zoom de la Figure 11 montrant en parme les sources de données biologiques disponibles sur le Web des Données

Une majorité des données biologiques disponibles l'ont été à l'aide de l'architecture Bio2RDF (Belleau et al. 2008). Le projet Bio2RDF consiste à fournir des outils open source et des bonnes pratiques permettant de faciliter la publication de données biologiques sur le Web des Données. Pour cela, la communauté Bio2RDF s'est appuyée sur les documents décrivant la transformation d'UniProt en RDF. Les données RDF créées sont rendues accessibles via un annuaire RDF Virtuoso proposant un point d'entrée SPARQL. Cette approche peut être rattachée aux approches d'entrepôt de données car les données sont

[7] http://dev.isb-sib.ch/projects/uniprot-rdf/migration.html

exportées de la base de données d'origine. De plus, grâce aux standards du Web Sémantique les annuaires RDF sont reliés entre eux sur le Web des Données. Bio2RDF a permis la conversion de plus de 40 bases de données créant en tout plus de 30 milliards de triplets (Nolin et al. 2010). L'autre solution utilisée par les biologistes pour rendre leurs données accessibles sur le Web des Données est D2R Server (Chris Bizer & Cyganiak 2006) qui propose également un point d'entrée SPARQL permettant de publier des bases de données relationnelles sur le Web des Données.

2.7.2. Les systèmes d'intégration du Web Sémantique

Nous appelons ici systèmes d'intégration du Web Sémantique, les systèmes d'intégration basés sur les standards du Web Sémantique.

Les technologies du Web Sémantique devenant mature, la communauté bioinformatique les a rapidement intégrées. YeastHub (K.-H. Cheung et al. 2005) est le premier cas d'utilisation Biologique de création d'un entrepôt de données basé sur des standards du Web sémantique. Les données sont stockées dans un annuaire RDF et peuvent être interrogées avec plusieurs langages de requêtes (RDF RQL, SeRQL et RDQL).

Un nombre grandissant de projets tentent de tirer profit des technologies du Web Sémantique pour intégrer des données biologiques (Ruttenberg et al. 2009). La création d'entrepôts de données basés sur les standards du Web Sémantique est aujourd'hui fréquente notamment grâce à l'utilisation des outils Virtuoso (OpenLink Software 2009) et D2RQ (Christian Bizer 2004). Ces derniers permettent notamment d'exporter des données contenues dans des bases de données relationnelles vers ces annuaires RDF puis d'y accéder via un point d'entrée SPARQL (Miles et al. 2010; Antezana, Blondé, et al. 2009; Lam et al. 2006).

Les technologies du Web Sémantique sont également utilisées pour créer des systèmes d'intégration basés sur l'approche de médiation. C'est le cas de la plateforme du TCGA (Deus et al. 2010) et très récemment de la plateforme iPlant (Goff et al. 2011).

2.8. Conclusion

La notion de Web Sémantique fut introduite en 1998 par Tim Berners-Lee et fut présentée comme « un ensemble d'applications connectées […] formant un Web de données basés sur une logique cohérente » (Berners-Lee 1998). Nous nous sommes appuyés sur la littérature pour montrer que le Web Sémantique arrive à maturité notamment grâce à la présence de formats standards et d'outils tels que les annuaires ontologiques ou les éditeurs

d'ontologie. Parmi les formats standards on retrouve RDF qui permet de définir des données présentes sur le Web, OWL qui est un langage sémantique permettant de publier et de partager des ontologies sur le Web ou encore SPARQL qui permet d'interroger des fichiers RDF.

Le Web Sémantique fournit une structure standard permettant de partager des données et de décrire explicitement leur sémantique, permettant ainsi de trouver, et intégrer des informations plus facilement. Le domaine de la biologie présente un fort besoin d'intégration de ses sources de données et s'est rapidement intéressé au Web Sémantique. Plus d'un dixième des sources disponibles sur le Web des Données sont des sources biologiques. Il existe également des plateformes d'intégration de type entrepôt de données ou médiateur basées sur des standards du Web Sémantique.

3. Les Services Web

3.1. Introduction

Idéalement, un biologiste devrait être capable de naviguer de manière transparente entre différentes applications et utilisant des données appropriées. En réalité, la large diversité de représentation de données, de schémas de bases de données ou encore de types de données utilisées en entrée et sortie d'une application, rend l'utilisation des ressources disponibles très difficiles. En réponse à cette complexité, les architectures à base de Services Web (SW) sont de plus en plus utilisées, permettant d'envelopper les outils et les données biologiques disponibles, et les rendant ainsi facilement accessible via le Web. Alors que le Web fournit une solution flexible permettant des interactions entre humain et machines, les SW permettent une interaction flexible entre machines dont le résultat peut être facilement utilisé par des humains. Plus d'un millier de SW biologiques sont disponibles sur le Web et ce nombre continue d'augmenter (Duncan Hull et al. 2005).

Nous allons dans un premier temps présenter le principe général des SW ainsi que les technologies associées. Nous présenterons ensuite dans une deuxième partie les raisons de la large utilisation de ces SW dans le domaine de la biologie. Dans une troisième partie nous présenterons les limites des SW puis dans la partie suivante nous présenterons comment ces limites peuvent être dépassées grâce à la combinaison des technologies des SW et du Web Sémantique. La cinquième partie détaillera les différentes approches de SW sémantiques réalisées dans le domaine de la biologie et la sixième partie présentera l'intérêt des annuaires de SW ainsi que certaines implémentations d'annuaires spécifiques au domaine biologique.

3.2. Le principe général des Services Web

Les SW fournissent un moyen standard de permettre l'interopérabilité entre applications logicielles, fonctionnant quel que soit la plateforme utilisée. Un SW est défini par le W3C comme étant « Un Service Web est un programme conçu pour assister l'interaction de plusieurs machines entre elles dans un réseau. Il possède une interface écrite dans un langage interprétable par une machine (le WSDL). D'autres systèmes interagissent avec le Service Web de la manière spécifiée dans la description de ce dernier, en utilisant des messages SOAP, généralement transmis par le protocole HTTP » (Booth et al. 2004).

Il existe plusieurs technologies derrière le terme SW. Les SW REST (Representational State Transfer) (Fielding 2000) sont basés sur l'architecture du Web et ses standards de base sont HTTP et URI. Les SW SOAP reposent sur les standards SOAP et WSDL afin de faciliter l'interopérabilité entre systèmes hétérogènes. Dans notre approche nous parlerons uniquement de SW de type SOAP.

3.2.1. Les principales technologies de Service Web

Les SW sont des applications modulaires ayant des interfaces de description pouvant être publiées, localisées ou encore invoquées à travers le Web. Initialement les SW ont été caractérisés par 3 technologies ressemblant fortement à HTML, HTTP et aux URIs de l'architecture du WWW (D. Fensel & C. Bussler 2002).

WSDL (Web Service Description Language) (Erik Christensen et al. 2001) permet de décrire la structure des SW en utilisant du XML. La deuxième version, appelée WSDL 2.0 (Chinnici et al. 2007), est devenu une recommandation du W3C en 2007. Un fichier WSDL est partagé en 6 éléments présentés dans la Figure 13.

SOAP (Gudgin et al. 2007) est un protocole permettant la transmission de messages entre des objets distants. Il défini une enveloppe sur le message lui-même afin de permettre son acheminement et son traitement. Il fournit également un modèle de données définissant les informations à transmettre.

UDDI (Universal Description, Discovery and Integration) (Clement et al. 2004) est une norme pour la mise en place d'annuaire de services permettant aux développeurs de logiciels de découvrir les SW existants.

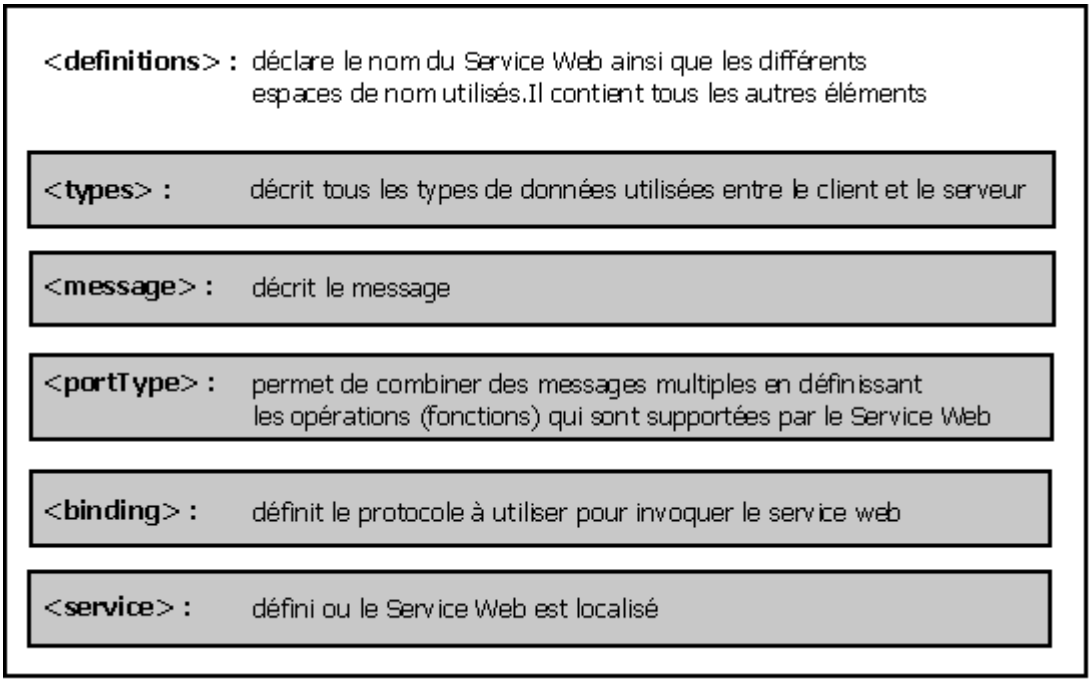


Figure 13 : Description des éléments d'un fichier WSDL

3.2.2. Les intérêts d'utilisation des Services Web dans la bioinformatique

Plusieurs milliers de SW sont référencés dans l'annuaire Biocatalogue (Bhagat et al. 2010) ; cela donne une indication du grand nombre de SW disponibles dans le domaine de la biologie. BioMart possède des SW permettant de réaliser des requêtes sur des bases de données relationnelles MySQL, Oracle ou Postgres (J. Zhang et al. 2011). Soaplab propose des SW servant d'adaptateurs pour EMBOSS (Martin Senger et al. 2009), l'EBI a développé plusieurs SW (Pillai et al. 2005; Mcwilliam et al. 2009), le NCBI a développé des SW pour les outils ENTREZ (Eric Sayers 2004), et des SW ont été développés pour KEGG (Kanehisa et al. 2002). De plus, beaucoup de laboratoires proposent un ou deux SW permettant d'accéder à leurs données. Cette adoption massive des SW peut s'expliquer par leurs caractéristiques (Sahoo et al. 2007):

- Un SW est une entité indépendante pouvant être invoquée à travers le Web quelle que soit l'application logicielle grâce à son interface bien définie. Cela permet aux SW d'être la plateforme idéale pour développer des outils, réalisant des traitements de données ou renvoyant des données, pouvant être facilement réutilisés par des biologistes, sans que ceux-ci aient à réaliser de grandes étapes d'implémentation.

- Les SW décrivent dans des documents XMLS leur interface, leurs entrées et sorties ainsi que les données échangées. La Figure 14 est une représentation schématique d'un SW.

Ce WS contient une opération implémentée dans un langage précis (brique blanche). La description des interfaces, entrées et sorties (brique verte et orange) permet d'encapsuler cette opération la rendant ainsi facilement réutilisable et plateforme indépendante. L'utilisation de la plateforme XML durant la totalité de leur cycle de vie, permet aux SW d'être compatibles avec une large gamme d'exigences.

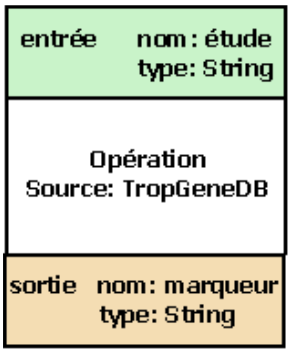


Figure 14 : représentation schématique d'un Service Web

- Plusieurs SW peuvent être enchainés permettant la réalisation de processus complexes de type flux de travaux. On parle alors de composition de services. Il est possible de réaliser une composition de SW permettant d'obtenir un plus gros jeu de données ou pour réaliser des taches complexes. Dans la partie A. de la Figure 15, tous les Services combinés réalisent la même opération sur plusieurs sources de données différentes. Les résultats renvoyés par le flux de travaux correspondront à la combinaison des résultats renvoyés par chaque SW. Dans la partie B. de la Figure 15, la sortie d'un SW est utilisée comme entrée d'un autre SW. Dans ce cas, une donnée devra être traitée par deux opérations avant d'être renvoyée. Taverna (D. Hull et al. 2006) est un logiciel permettant à des bioinformaticiens de créer des flux de travaux de manière intuitive, à l'aide d'une interface graphique.

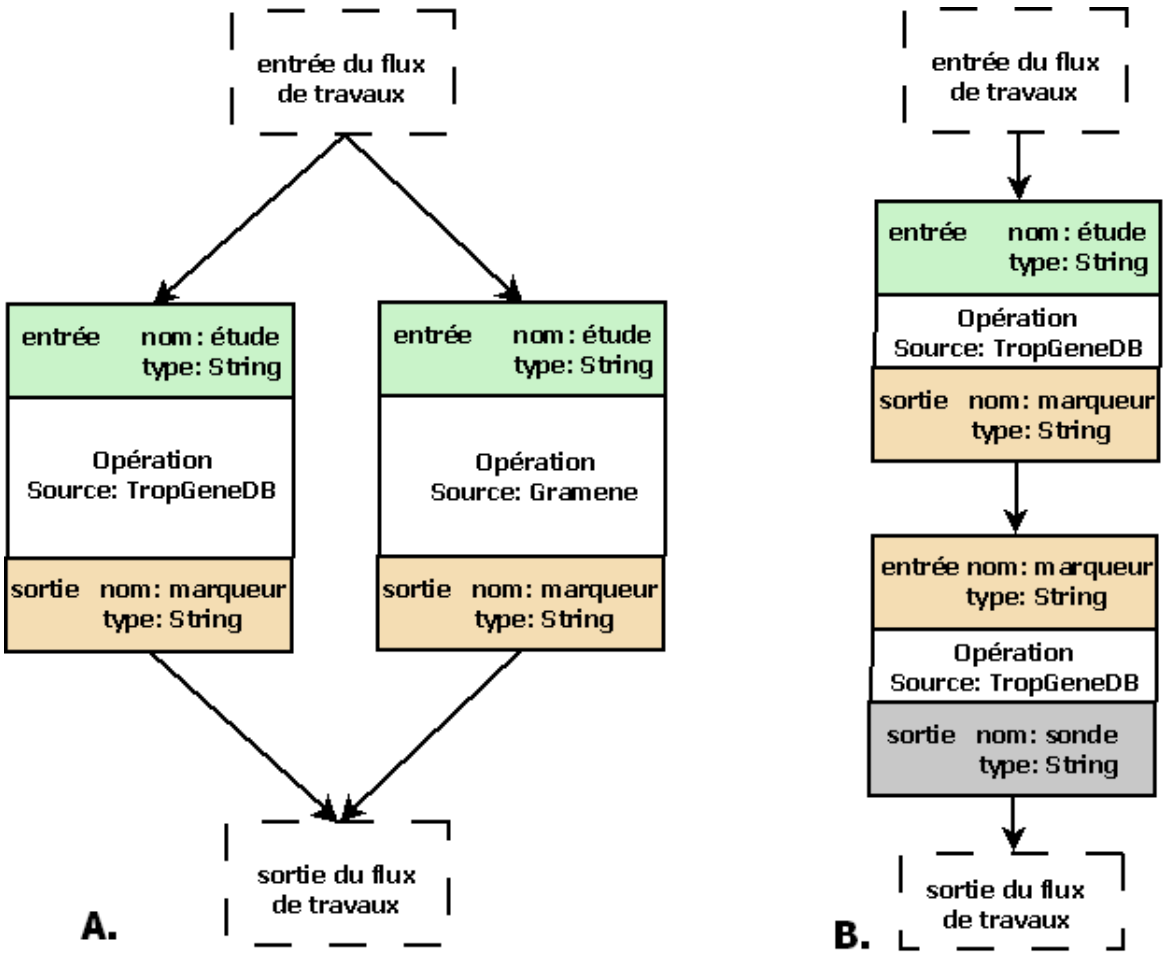


Figure 15 : Composition de Services Web

3.2.3. Le principe d'utilisation des Services Web

L'architecture d'utilisation d'un SW est présentée dans la Figure 16. Cette architecture est composée de 3 acteurs ; le fournisseur, l'annuaire et le client. L'utilisation est réalisée en 6 étapes. Dans un premier temps, le fournisseur enregistre son SW dans un annuaire. L'annuaire va dans un deuxième temps récupérer des informations du fichier WSDL pour les publier. Les étapes 3 et 4 consistent respectivement à ce qu'un client découvre un SW puis récupère les informations nécessaires à son utilisation. Les 2 dernières étapes correspondent à l'utilisation proprement dite du SW. Le client va l'invoquer (étape 5) puis récupérer les résultats qu'il souhaite (étape 6).

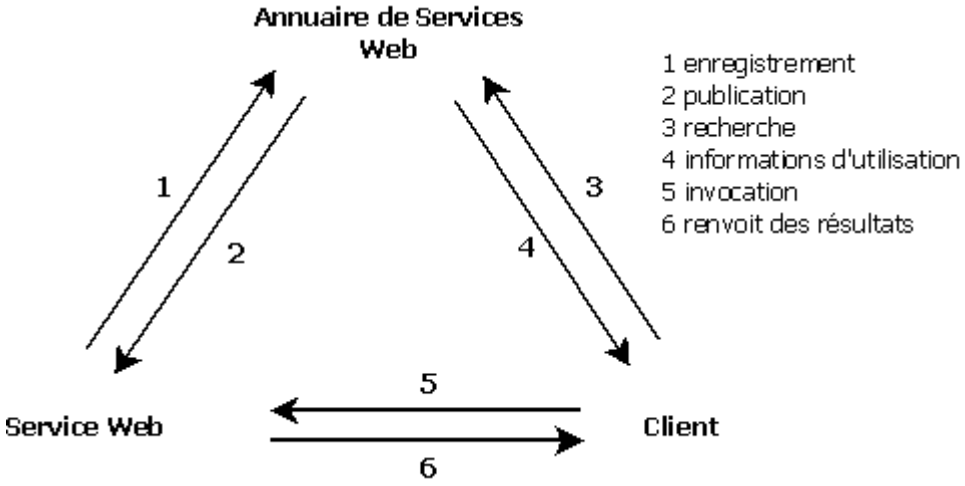


Figure 16 : Architecture d'utilisation d'un Service Web

3.3. Les limites des Services Web classiques

Les interfaces des SW permettent de rechercher et de renvoyer facilement des données. De plus, il est possible de détecter ces SW en utilisant des annuaires spécialisés. Cependant, étant donné l'objectif de réaliser une interopérabilité basée sur le Web entre applications ad hoc, l'approche des SW souffre de plusieurs problèmes (Törmä et al. 2008).

Couverture insuffisante des propriétés des SW : Les descriptions WSDL actuelles permettent de décrire l'interface pouvant être utilisée pour invoquer un SW. WSDL ne supporte pas de description des propriétés fonctionnelles (coût, qualité, disponibilité), la composition ou le comportement interne d'un SW. Cela signifie que certaines propriétés, permettant la détection ou la composition automatique de SW, doivent être décrites avec d'autres moyens que le WSDL.

Terminologie non structurée de description de SW : WSDL permet une description structurale des SW mais celle-ci est purement syntaxique. Cela peut être comparé à la documentation d'une librairie d'un langage de programmation dans le sens ou les opérations et les types, définis dans le WSDL, révèlent autant sur le sens du SW que le nom des procédures et des types dans un langage de programmation. WSDL ne permet pas d'ajouter d'informations sémantiques (Carole Goble & De Roure 2008). Ainsi, l'utilisation d'un SW ou la composition de SW nécessite la présence d'un développeur logiciel lisant la documentation et décidant quel SW utiliser et comment l'utiliser.

Ces limitations posent un problème à la détection automatique de SW, mais également à leur détection manuelle. En effet, étant donné la très grande quantité de SW disponibles

dans le domaine biologique, l'absence d'informations sémantiques notamment sur le type d'entrées et de sorties peut rendre la recherche d'un SW très difficile.

3.4. Les Services Web Sémantiques

Le développement de technologies Services Web Sémantiques (SWS) a pour but d'enrichir les technologies existantes des SW à l'aide des capacités apportées par le Web Sémantique. L'utilisation de SWS doit permettre de détecter automatiquement les SW disponibles, de réaliser de la composition automatique de SW et de raisonner sur le but précis auquel doit répondre un agent logiciel. Deux ontologies ont été proposées, OWL-S et WSMO :

- **OWL-S** (Web Ontology Language for Services) (D. Martin et al. 2004) tente de combiner les technologies de représentation du Web Sémantique OWL et RDF, avec les standards des SW comme WSDL. Techniquement OWL-S est une ontologie divisée en 3 sous ontologies *Profile*, *Process* et *Grounding* décrivant respectivement ce que fait un SW, comment il fonctionne et comment l'utiliser.
- **WSMO** (Web Service Modeling Ontology) (Bruijn et al. 2005) est basé sur WSMF (D. Fensel & C. Bussler 2002), dont le but est de découplé au maximum les différents composants du SW. La partie principale de la description de SW dans WSMO est constituée de pré-conditions, post-conditions, suppositions et effets.

Les ontologies OWL-S et WSMO sont toutes les deux complexes et extrêmement expressives. Ces standards contiennent des caractéristiques qui ne sont pas nécessaires à la communauté bioinformatique (Duncan Hull et al. 2005). **SAWSDL** (Semantic Annotation for WSDL) (Farrell & Lausen 2007) est une recommandation du W3C qui fournit une approche plus légère permettant de créer des SWS. Il permet d'ajouter des annotations sémantiques, c'est-à-dire de rajouter des informations qui identifient ou définissent un concept d'un modèle sémantique afin de décrire une partie du document, dans différentes partie d'un fichier WSDL, notamment au niveau de la structure des entrées et sorties, de l'interface et des opérations.

3.5. Les Services Web Sémantiques en Biologie

En parallèle des approches développées dans le domaine informatique, des projets bioinformatiques permettant la création de SWS ont vu le jour.

- **BioMoby** (The BioMoby Consortium 2008) fut le premier projet bioinformatique ayant pour but de créer des SW biologiques contenant de la sémantique. Cette sémantique est ajoutée aux SW en utilisant une ontologie de types de données, pour définir les entrées et sorties des SW, ainsi qu'une ontologie de services, permettant de définir le type d'action réalisée par le SW. BioMoby permet de générer automatiquement la structure d'un SW sous la forme d'un squelette Java, contenant notamment les méthodes et les objets à utiliser, qu'un bioinformaticien doit ensuite implémenter puis déployer dans un annuaire spécifique appelé « Moby Central ». En parallèle du développement de BioMoby, le projet ^{my}Grid a développé le logiciel Taverna permettant de créer des flux de travaux de manière graphique. Taverna represente un SW sous la forme d'une brique graphique, il est possible de réaliser une composition de SW simplement en reliant graphiquement 2 briques entre elles. Un plugin (Kawas et al. 2006) pour Taverna a été développé, lui permettant de composer des SW BioMoby. Actuellement, plus de 750 SWS BioMoby sont disponibles.
- Le projet **SSWAP** (Simple Semantic Web Architecture and Protocol) (Damian Gessler, G. Schiltz, et al. 2009) fut initialement créé comme une branche parallèle du projet BioMoby appelée Sémantique Moby. Il est basé sur les standards OWL et RDF pour construire un système permettant une description, une découverte et une invocation basées sur une sémantique robuste. Cela permet d'utiliser des ontologies partagées existantes et de pouvoir raisonner. Il est également possible de créer sa propre ontologie OWL et de l'utiliser pour créer ses propres SWS. SSWAP est basé sur une ontologie OWL composée de 5 classes permettant de décrire une ressource Web, l'entrée et la sortie du SWS, la structure des données ainsi que le fournisseur de données. Le projet SSWAP est utilisé pour créer les adaptateurs de la partie médiation de la plateforme iPlant (Nelson et al. 2010).
- **SADI** (Semantic Automated Discovery and Integration) (M. D. Wilkinson et al. 2009) est le projet bioinformatique de création de SWS le plus récent. Il a été mis en place pour dépasser les limites du projet BioMoby. SADI, contrairement à SSWAP et BioMoby, ne veut pas créer un nouveau message d'invocation et de réponse, de manière à ne pas être spécifique à une architecture donnée. Le but de SADI est de permettre la création de SWS s'intégrant naturellement dans le Web sémantique. L'interface d'un SW est définie à l'aide de deux classes OWL représentant l'entrée et la sortie du Service. Le SWS consomme des individus de la classe d'entrée et renvoie des individus de la classe de sortie. Dans un SWS SADI l'URI des individus d'entrée

et de sortie doivent être les mêmes. Cela signifie que l'entrée et la sortie de ces services sont explicitement reliées par une suite de prédicats RDF (M. Wilkinson et al. 2010).

- **BioXSD** (Kalas et al. 2010) est un projet ayant pour objectif de développer un format d'échange de données biologiques se voulant standard et canonique. BioXSD se présente sous la forme d'un fichier XMLS qui définie la syntaxe de séquences biologiques, d'annotation de séquences, d'alignement et de référence vers des ressources. Mais BioXSD ne s'arrête pas à une description syntaxique des données. Il intègre également de la sémantique, apportée par l'ajout d'annotation vers l'ontologie EDAM sous la forme de balises SAWSDL. BioXSD a été utilisé pour annoter les entrées et sorties de SWS pouvant être utilisés dans des flux de données. Le XMLS BioXSDL n'est pas un standard fermé: une nouvelle version contenant des annotations vers plusieurs bioontologies est en cours de développement.
- **BioCatalogue** (Bhagat et al. 2010) est un annuaire de SWS spécifique au domaine de la biologie. Il permet d'enregistrer des SWS facilement via une interface Web, puis de rajouter des annotations sémantiques, sous la forme d'étiquettes, à différents niveaux du Service. Il est par exemple possible de rajouter une étiquette générale décrivant le SWS, des étiquettes décrivant les opérations ou encore les entrées et sorties de chaque opération. Ces étiquettes peuvent être utilisées par un consommateur de données, de manière à détecter plus facilement le SW qu'il souhaite utiliser. Il s'agit de l'annuaire de SW le plus utilisé dans le domaine de la biologie, il recense actuellement plus de 2000 SWS rendus disponibles par plus de 150 fournisseurs de données.

3.6. Conclusion

Les SW SOAP sont basés sur 3 technologies : WSDL permettant de décrire le SW, SOAP permettant la transmission de messages et UDDI permettant de créer des annuaires regroupant différents SW. Les SW sont largement utilisés en biologie car ils peuvent être facilement réutilisés par différents consommateurs de données sans qu'ils n'aient à se soucier de la plateforme d'implémentation utilisée. Il est également possible de réaliser une composition de SW existants, de manière à créer des flux de travaux complexes. Cependant, les SW sont souvent décrits uniquement de manière syntaxique; il n'est donc pas possible de déterminer la sémantique associée sans intervention humaine. Pour répondre à ce problème il est nécessaire de combiner les technologies des SW et du Web Sémantique.

Plusieurs approches ont été définies dans le domaine informatique comme OWL-S et WSMO. Il s'agit d'approches dont l'utilisation est lourde et dont toutes les caractéristiques ne sont pas utilisées en bioinformatique. C'est pour cela que plusieurs projets bioinformatiques tels que BioMoby, SSWAP, SADI ou encore BioXSD ont traité de la création de SWS.

L'utilisation de SWS dans le domaine de la bioinformatique est très encourageante car les Services Web fournissent une interface de programme et évite de parcourir des interfaces web pour récupérer des informations réparties (Lincoln Stein 2002), et les descriptions sémantiques favorisent leur découverte et leur composition.

4. Les correspondances entre bases de données et ontologies

4.1. Introduction

Au cours des dernières années le Web Sémantique est passé du statut de vision futuriste abstraite à la réalité de plus en plus proche d'un web global de données reliées et dont la sémantique est définie explicitement. Des langages et technologies standards ont été proposés et sont en constante évolution de manière à construire les bases pour cette prochaine génération du Web. Des applications dans le monde réel proposent déjà un avant goût des avantages du Web Sémantique dans différents domaines, notamment dans le domaine de la biologie. La participation des bases de données et leur rôle dans cette évolution du Web est effective depuis le tout début de la conception du Web Sémantique non seulement parce qu'il a été initialement comparé à « une base de données globale » (Berners-Lee 1998) mais aussi car il tire avantage de la grande expérience et de la maturité du monde des bases de données. Au départ, les systèmes de base de données ont été considérés par la communauté du Web Sémantique comme un excellent moyen de stockage pour les ontologies (Dave Beckett & J. Grant 2003) en raison des bénéfices connus et justifiés de leurs performances.

Les ontologies souffrent d'un problème d'extensibilité. Le fait de stocker une grande quantité de données dans un unique fichier est une pratique peu efficace. C'est pourquoi les ontologies devraient se limiter à définir le contenu plutôt que de le contenir. En attendant, le web sémantique n'est pas destiné à être une technologie candidate pour l'ingénierie du web. Il s'agit d'une addition aux pratiques actuelles plutôt qu'une substitution. La clé pour relier des données à l'aide de significations sémantiques formelles est l'inclusion d'une médiation entre les composants des bases de données relationnelles et les ontologies (Konstantinou et al. 2008).

4.2. Qu'est ce qu'un mapping entre bases de données et ontologies

Un mapping (mise en correspondance) est décrit dans le MOF 2.0 (Object Management Group 2002) comme étant la spécification d'un mécanisme permettant de transformer les éléments d'un modèle, conformément à un métamodèle vers les éléments d'un autre modèle qui est conforme à un autre métamodèle (pouvant être le même). Un mapping est exprimé à l'aide d'associations, de contraintes et de règles auxquelles on assigne des paramètres durant l'étape du mapping. La problématique du mapping peut être vue comme un cas particulier d'un problème rencontré en intégration des données, le matching de schémas. Dans le matching de schéma, deux sous ensembles de données sont présents: le schéma source ou schéma local et le schéma cible ou schéma global. Le but du matching est de proposer une interface de requête commune pour les données issues des sources hétérogènes.

Dans notre cas, nous allons nous intéresser à un type précis de mapping, le mapping entre une base de données et une ontologie qui est couramment vu comme une problématique spécifique à l'intégration de données. Théoriquement un système d'intégration de données est un triplet <G,S,M> où G est le schéma global, S est le schéma source et M est l'ensemble des déclarations permettant de relier les éléments du schéma source aux éléments du schéma global. Dans notre cas nous considérons le schéma d'une base de données comme un schéma local et le schéma ontologique comme le schéma global. Le problème général de l'intégration de données est donc réduit ici à la recherche de correspondances entre un ensemble de données relationnelles et ontologiques.

Le but des outils de mapping entre ontologie et base de données est de donner accès aux données contenues dans une base de données en utilisant le schéma d'une ontologie. Ce mapping est perçu différemment que l'on se trouve dans le domaine du Web Sémantique ou des bases de données relationnelles. Du point de vue du web sémantique, ce mapping permet de faire correspondre les individus d'une classe à toute combinaison d'un sous ensemble de données possibles. Cela permet de d'augmenter considérablement les capacités de stockage d'une ontologie. Du point de vue des bases de données, ce mapping permet d'enrichir la structure du schéma de la base. Cela permet d'inclure de la logique de description dans le schéma et ainsi répondre à des requêtes d'un niveau sémantique plus élevé.

4.3. Les bases de données et les bases de connaissances

Les difficultés de mapping viennent de l'hétérogénéité entre les bases de données et les ontologies. Par exemple, une base de données ne fournit pas une sémantique formelle sur les données qu'elle contient, au contraire d'une ontologie. De plus, un schéma de base de données n'est, à l'origine, pas fait pour être partagé, ni réutilisable. Il est modélisé pour définir une base de données spécifique. L'ontologie est quand à elle prévue pour être partagée et réutilisée. Une autre grande différence entre les ontologies et les bases de données est l'approche utilisée pour le développement. En effet, le développement d'une ontologie est un effort collaboratif nécessitant la présence d'un grand nombre de personnes. Le développement du schéma d'une base de données est quand à lui rarement le résultat d'un travail d'équipe.

Les bases de données et les bases de connaissances sont similaires dans le sens ou elles sont toutes les deux utilisées dans le but de maintenir des modèles et des données de plusieurs domaines de discussion (Borgida et al. 2003). Il y a cependant une grande différence entre les deux, car les bases de données manipulent des modèles larges et persistants contenant des données relativement simples, alors que les bases de connaissance contiennent moins de données mais ces données sont plus complexes. Les bases de connaissances ont la possibilité de renvoyer des réponses sur le modèle alors qu'elles n'ont pas été demandées explicitement. Ainsi, mapper une base de connaissance et une base de données permet d'augmenter la possibilité de renvoyer implicitement de la connaissance de la base de données.

Cela est rendu possible par l'utilisation de la terminologie décrite dans la TBox. Le schéma d'une base de données peut être comparé à une TBox d'une base de connaissances, et les instances présentent dans une base de données peuvent être comparées à la ABox d'une base de connaissances.

Cependant, il y a de grandes différences entre une base de données et une base de connaissance. La plus importante de ces différences est l'expressivité. En effet, là où un schéma ontologique permet d'exprimer un grand nombre de relations, le schéma relationnel d'une base de données permet uniquement d'exprimer des relations IS-A. Ces relations ne fournissent pas assez de caractéristiques pour permettre de déterminer des relations complexes entre les données.

Une autre différence importante est le type de sémantique présente sur chaque type de schéma. Un schéma de base de données est régit par ce qui est appelé "l'hypothèse du monde

fermé". Cela signifie que tout ce qui n'a pas été explicitement déterminé comme vrai dans un tel schéma est considéré comme faux. Ainsi, si une valeur nulle est présente dans le champ d'une table, l'absence d'information sera considéré comme volontaire. Par exemple si un champ appelé *EstUnADNc*, comporte des données de types booléen définissant si la donnée correspond à un ADNc, et si un enregistrement comporte la valeur nulle pour ce champ, cela sera interprété comme le fait que cet enregistrement ne correspond pas à un ADNc.

Au contraire, une base de connaissance suit l'hypothèse appelée "hypothèse du monde ouvert". Une base de connaissance peut renvoyer 3 types de résultat: vrai, faux ou ne peux être déterminé. Dans ce cas, le manque d'information n'implique pas de considéré le résultat comme faux. Dans une base de connaissance, l'absence d'information permettant de savoir si un enregistrement correspond à un ADNc renverra que l'on ne peut déterminer s'il s'agit d'un ADNc.

4.4. Des motivations variées

Il est important d'identifier les différentes motivations et les problèmes qu'impliquent l'interaction entre les bases de données relationnelles et les technologies du Web Sémantique. Cela permet de séparer de façon cohérente les différents challenges et buts de ces approches. Dans cette partie nous allons détailler plusieurs bénéfices pouvant être tirés de l'interconnexion de bases de données et d'ontologies.

L'annotation sémantique de pages web dynamiques : des pages HTML peuvent être annotées sémantiquement à l'aide de termes issus d'ontologies. Cela rend leur contenu accessible par des agents logiciels et des Services Web. Cependant ce scénario ne fonctionne pas correctement pour les pages Web créées dynamiquement à partir de bases de données. Ces pages Web dynamiques représentent la plus grande partie des données disponibles sur le Web, formant ce qui est appelé le Web profond (Cyga 2005) qui n'est pas accessible aux agents logiciels ou aux moteurs de recherche. Une solution possible à ce problème serait d'annoter directement le schéma de ces bases de données plutôt que les pages Web, de manière à révéler la structure de la base de données. Cette annotation consiste en un ensemble de correspondances entre les éléments d'un schéma de base de données et des ontologies déjà existantes de manière à fixer le domaine du contenu de ces pages dynamiques.

L'intégration de bases de données hétérogènes : une hétérogénéité est présente entre deux systèmes de base de données quand ils utilisent des infrastructures logicielles ou matérielles différentes, quand ils suivent des conventions syntaxiques ou des modèles de représentation différents, ou quand ils interprètent différemment des données identiques ou similaires (A. P. Sheth & Larson 1990). Dans un cas classique d'intégration de bases de données, un ou plusieurs schémas conceptuels sont utilisés pour décrire le contenu de chaque source. Les requêtes sont posées sur un schéma conceptuel global et, pour chaque source de données, un adaptateur permet de reformuler la requête et ainsi renvoyer les données souhaitées. L'intégration basée sur les ontologies utilise les ontologies à la place d'un schéma conceptuel et définie des correspondances entre l'ontologie et chaque source. Ainsi, la découverte et la représentation de mappings entre des schémas de base de données relationnelle et des ontologies constitue une part importante du scénario d'intégration de bases de données hétérogènes.

Accès aux données, basé sur une ontologie : cette approche suppose que le lien entre une ontologie et une base de données agit comme une couche intermédiaire entre l'utilisateur et les données stockées. L'ontologie fournit une abstraction du contenu de la base de données, permettant aux utilisateurs de formuler des requêtes de haut niveau de description sur un domaine d'intérêt. L'accès aux données basé sur une ontologie peut être vu comme un adaptateur dans un système d'intégration car il cache les détails de la source en transformant les requêtes posées sur le schéma conceptuel en requêtes interrogeant la source de données d'origine. L'avantage principal de cette approche est la possibilité d'interroger une base de données sans avoir à exporter son contenu.

Génération de masse de données sur le Web des Données : une base de données relationnelle est le moyen de stockage de données le plus populaire. L'extraction, si possible automatique, de données contenues dans les bases de données vers RDF est une solution pour créer une masse considérable de données accessibles sur le Web des Données. Le terme de mapping entre bases de données et ontologies est également utilisé dans la littérature pour décrire ce type d'approche.

Aide à la création d'ontologie : le processus de développement d'ontologie en partant de zéro est une tache difficile, consommatrice de temps et sujet aux erreurs. Les bases de données relationnelles sont des sources d'information structurées et leurs schémas est crée en suivant des méthodologies standard. Elles constituent une source significative et fiable d'un domaine de connaissance. Elles peuvent donc être utilisées pour créer des ontologies riches en réunissant les informations issues de leurs schéma, contenu, requêtes et procédures, du moment qu'un expert supervise le processus et enrichit le résultat final.

Définition du sens d'un schéma relationnel : la création d'une base de données débute par la création d'un modèle conceptuel. Ce modèle conceptuel n'est souvent pas conservé suite à la phase de création du schéma relationnel. Cela conduit à des bases de données ayant perdu l'intention originale de leur concepteur et devenant compliqué à étendre ou réutiliser dans un autre modèle logique. L'établissement de correspondances entre un schéma relationnel et une ontologie permet de maintenir la signification originale à l'aide d'un modèle conceptuel expressif.

4.5. Les types d'approches existantes

La problématique de mapping entre base de données relationnelles et ontologies peut être dépassée en utilisant un grand nombre d'approches différentes. De manière à détailler et décrire ces approches, nous avons reproduit la Figure 17, issue de (Spanos et al. 2011), qui représente une taxonomie de classification des approches existantes. Il est important de souligner que chaque classe de cette taxonomie est hétérogène et peut ainsi contenir des outils ayant des motivations variées.

La différence la plus importante entre les approches de mapping est basée sur l'existence ou non d'une ontologie. C'est le critère utilisé pour la première classification des approches. Dans le cas des approches utilisant une **ontologie déjà existante**, l'ontologie mappée à la base de données devrait modéliser un domaine compatible avec le modèle de la base de données. Dans ce cas, les ontologies sont sélectionnées par un utilisateur expert, conscient de la nature des données contenues dans la base de données.

Dans le cas de méthodes créant une **nouvelle ontologie**, l'existence d'une ontologie de domaine n'est pas obligatoire. Cette approche peut être réalisée même si aucune ontologie

couvrant le domaine de la base de données existe, ou si l'utilisateur n'est pas familier avec le domaine de la base de données, et compte sur le processus de mapping pour déterminer la sémantique du contenu de la base de données.

La classe des approches créant une nouvelle ontologie peut être divisée en 2 classes différentes. D'un côté il y a des approches créant une **ontologie dirigée structurellement**. Ces ontologies sont créées en utilisant des classes et des propriétés prédéfinies (ex :Table, Column, hasColumn) de manière à décrire uniquement la structure de la base de données en ignorant la sémantique du modèle relationnel. Ces approches sont souvent automatiques et dépendent uniquement du schéma de la base de données. De l'autre coté il existe des approches permettant la création d'une **ontologie dirigée sémantiquement**. Le développement de cette ontologie consiste dans un premier temps à détecter des concepts et des relations entre ces concepts, en se basant sur le schéma relationnel. Le phase de création de l'ontologie se basera sur ces concepts et relations pour créer les classes et relations ontologiques correspondantes.

La classe des approches basées sur la création d'une ontologie dirigée sémantiquement peut également être divisée en 2 classes différentes. Les approches **utilisant l'ingénierie inverse** essayent de retrouver le schéma conceptuel à partir du schéma relationnel et utilisant ces informations pour créer la nouvelle ontologie. De l'autre coté, les approches **n'utilisant pas l'ingénierie inverse** appliquent des règles de transformation basiques du modèle relationnel vers le modèle RDF et nécessitent l'intervention d'un expert pour définir des mappings complexes enrichissant le modèle de l'ontologie créée.

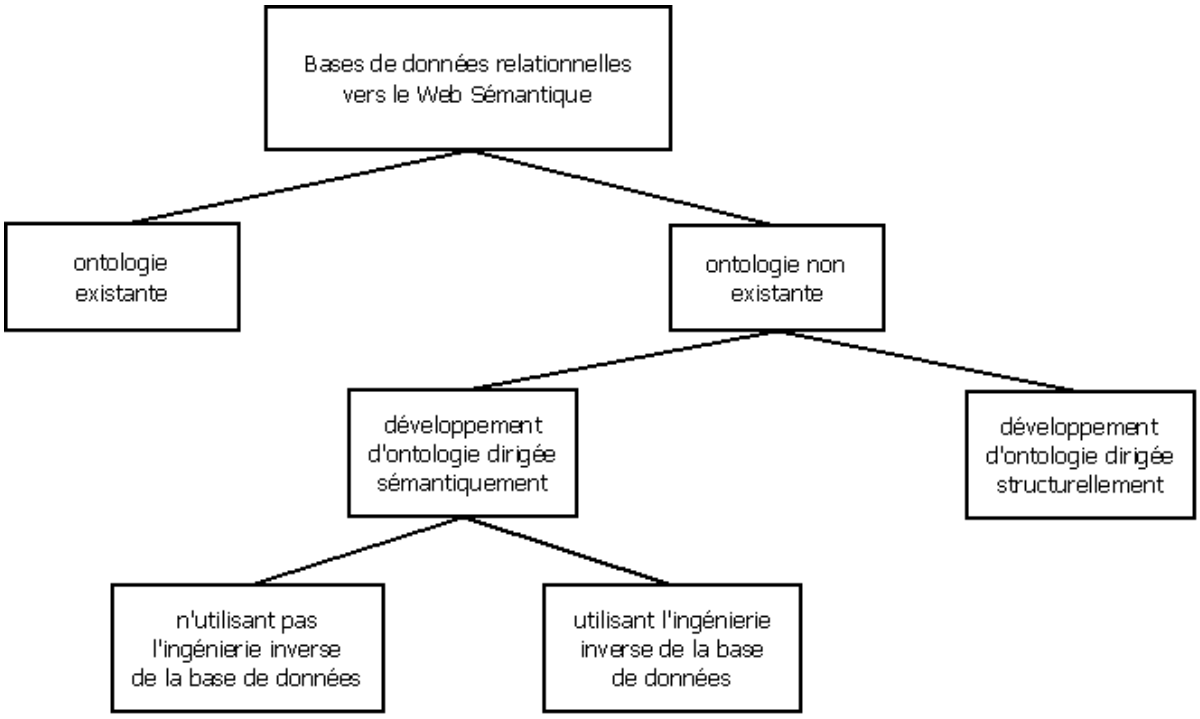


Figure 17 : Taxonomie simple des approches existantes (Spanos et al. 2011)

4.6. Détails de projets existants en informatique

Dans cette partie nous détaillerons uniquement des projets permettant de définir des mappings entre ontologies et bases de données. Nous ne détaillerons pas les approches permettant le simple stockage d'ontologies dans une base de données comme c'est le cas pour 3Store(S. Harris & Gibbins 2003), Corese (Corby et al. 2004), Sesame(Broekstra et al. 2001), Kowari (Wood 2005) ou encore Instance Store(Horrocks et al. 2004). Nous ne présenterons pas une liste exhaustive des projets existants mais une sélection des approches nous semblant les plus abouties, tout en balayant l'intégralité des classes de la taxonomie présentée précédemment.

MOMIS (Sonia Bergamaschi et al. 2005; S Bergamaschi 2001) est un outil permettant de gérer l'intégration de sources d'informations hétérogènes, correspondant à la classe ontologie existante de notre taxonomie. Il permet un accès intégré à des données stockées dans des bases de données ou dans des fichiers systèmes. Il utilise ODL_{I3}, un langage orienté objet dérivé du standard ODMG (Object Database Management Group), de manière à représenter la sémantique du schéma de la source. Les composants principaux de MOMIS sont :

- les adaptateurs permettant de gérer toutes les sources locales,

- un médiateur comprenant un constructeur d'ontologie et un gestionnaire de requêtes,

- un outil réalisant la classification des classes ODL_{I3} locales pour la synthèse des classes ODL globales,

- un outil inférant de nouvelles relations entre les classes ODL_{I3} locales et contribuant à la génération d'un thésaurus commun.

La transformation du schéma des sources vers les langages ODL_{I3} est basé sur les règles suivantes: (1) le nom d'une relation correspond à une classe ODL_{I3}, (2) pour chaque attribut d'une relation, un attribut est défini dans la classe ODL_{I3} correspondante.

D2R MAP (Christian Bizer 2003) permet de réaliser un mapping déclaratif entre une ontologie et une base de données. Il s'agit d'une approche de création d'ontologie dirigée sémantiquement sans ingénierie inverse. D2R Map est basé sur une syntaxe XML et constitue un langage permettant de gérer des correspondances entre des concepts ontologiques et des éléments de base de données. Il autorise des mappings complexes en employant des déclarations SQL dans les règles du mapping, sans avoir à modifier le schéma de la base de données. D2R MAP peut manipuler des tables ayant une structure hautement normalisée et où les instances sont réparties sur plusieurs tables. Cependant, il ne permet pas de mapper les bases de données peu structurées en raison de son expressivité limitée.

D2RQ (Christian Bizer 2004) est basé sur les mêmes concepts que D2R MAP, mais avec syntaxe différente. Il s'agit donc également d'une approche de création d'ontologie dirigée sémantiquement sans ingénierie inverse. L'ontologie D2RQ est implémentée comme un graphe RDF, contenant le schéma d'une ou plusieurs bases de données relationnelles. Il s'agit d'un graphe RDF virtuel permettant de se connecter aux bases de données relationnelles et de lire leur contenu. En utilisant D2RQ, une application peut interroger une base de données non RDF en utilisant les langages de requêtes SPARQL ou RDQL. D2RQ permet de réécrire ces requêtes en SQL interrogeant directement la base de données relationnelle correspondante. Le résultat de ces requêtes SQL est ensuite transformé en triplets RDF pouvant être renvoyé à l'application initiale. La création du graphe virtuel RDF est réalisée semi-automatiquement. Une première étape automatique permet de récupérer le schéma de la base de données, puis la seconde étape consiste

à annoter manuellement les éléments du schéma à l'aide de concepts ontologiques. Comme les annotations sémantiques sont réalisées manuellement, le graphe virtuel RDF doit être refait à chaque modification du schéma de la source.

R2O (Relational To OWL) (Barrasa et al. 2004) est un langage extensible et déclaratif basé sur le langage XML, nécessitant l'existance d'une ontologie. Il permet de définir les mappings entre des schémas de base de données et des ontologies implémentées en RDF ou OWL. Comme D2R MAP, R2O permet de définir explicitement des correspondances entre les deux modèles. R2O est un langage de haut niveau qui peut être utilisé avec tout type de base de données s'appuyant sur le standard SQL. Un mapping R2O définit la façon de créer des instances dans l'ontologie en fonction des données stockées dans la base de données. L'approche consiste à créer un fichier de description de mappings en utilisant R2O, avec toutes les correspondances entre les éléments de la base de données et ceux de l'ontologie. Ce fichier sera ensuite utilisé pour peupler l'ontologie, c'est-à-dire exporter les données contenues dans la base de données et les intégrer dans l'ontologie en temps qu'instances de classes.

Depuis 2006 R2O est couplé au moteur ODEMapster (Barrasa et al. 2006). Ce moteur permet d'utiliser un fichier de mapping R2O pour accéder aux données dans la source d'origine à l'aide de requêtes créées au format ODEMQL.

L'architecture **Ontograte** (Dou & LePendu 2006) permet de transformer un schéma relationnel en une représentation ontologique. Elle peut être classifiée comme une architecture de création d'ontologie dirigée sémantiquement sans ingénierie inverse. Cette approche regroupe une représentation de schéma basé sur les ontologies, une inférence en logique de premier ordre et des adaptateurs SQL.

Relational.OWL (de Laborda et al. 2005) est un langage permettant d'extraire la structure d'une base de données relationnelle et de la transformer en une ontologie RDF/OWL. Dans notre taxonomie, il correspond à la classe de création d'ontologie dirigée structurellement. Les données sont représentées comme des instances de l'ontologie spécifique à la source de données. Relational.OWL utilise les techniques fournies par OWL pour créer des classes comme

Table ou *Column*, et préciser les relations possibles entre ces classes (hasColumn). L'ontologie est créée automatiquement.

ROSEX (Curino et al. 2009) utilise une version légèrement modifiée de l'ontologie Relational.OWL de manière à représenter le schéma relationnel d'une base de données comme une ontologie. Il n'utilise plus la couche de représentation de données présente dans Relational.OWL. et peut être considéré comme un outil de création d'ontologie dirigée sémantiquement avec ingénierie inverse. L'ontologie représentant le schéma de la base de données est mappé à une ontologie de domaine qui a été créée automatiquement en se basant sur le schéma de la base de données. Ce mapping est utilisé pour réécrire des requêtes SPARQL, exprimées sur l'ontologie de domaine, vers des requêtes SQL exprimées sur la base de données relationnelle.

OpenLink Virtuoso Universal Server est une plateforme d'intégration existant dans une version commerciale et une version open-source. Cette plateforme permet de créer une vue RDF à partir d'une base de données relationnelle en s'appuyant sur les caractéristiques de ces vues RDF (OpenLink Software 2009), qui offrent les mêmes caractéristiques que D2RQ. Tout comme D2RQ il s'agit d'une approche de création d'ontologie dirigée sémantiquement sans ingénierie inverse. Virtuoso supporte un mode automatique et un mode manuel. La partie automatique consiste à créer une ontologie RDFS. Ensuite, le langage de méta-schéma Virtuoso peut être utilisé pour définir des mappings complexes. Le langage Virtuoso possède la même expressivité que le langage D2RQ.

MASTRO (Calvanese et al. 2011) est un outil permettant de définir manuellement des mappings entre une base de données relationnelle et une ontologie. Il s'agit d'une approche nécessitant la présence d'une ontologie. MASTRO permet un accès aux données basé sur l'ontologie. Il comporte un algorithme permettant de reformuler une requête SPARQL en une requête SQL. MASTRO est également complémenté avec le plugin ODBA pour l'éditeur d'ontologie Protégé. Ce plugin permet de définir des mappings de type GAV entre une base de données et une ontologie à l'aide d'une interface graphique (Poggi et al. 2008).

MAPONTO (An et al. 2006) est un outil semi automatique aidant un utilisateur expert à découvrir des relations sémantiques entre un schéma de bases de données et une ontologie, en lui proposant des correspondances entre les attributs d'une relation et des propriétés. Il est classifié comme un outil nécessitant l'existance d'une ontologie. MAPONTO visualise le schéma relationnel comme un graphe ou les nœuds, représentant les concepts, sont reliés par des clés étrangères. MAPONTO détecte les propriétés de type de données associées à chaque classe de l'ontologie, et tente de trouver une table de la base de données dont les colonnes correspondent aux mêmes types de données. Si une telle correspondance est trouvée, elle est proposée à l'utilisateur. Ce dernier pourra choisir de valider les correpondances, permettant l'extraction de mappings de type LAV pouvant être utilisées dans le cadre d'intégration de bases de données.

L'approche détaillée par **Tirmizi et al** (Tirmizi et al. 2008) propose un ensemble de règles permettant de créer une ontologie à partir d'un schéma relationnel exprimé en SQL. Les règles décrites permettent de réaliser une ingénierie inverse du schéma relationnel pour retrouver des informations du schéma conceptuel. Ces règles permettent notamment d'identifier des relations d'héritages, des relations binaires, des relations n-aires, des contraintes (non nul, unique) et la transformation du type de données.

Le Tableau 1 ci-dessous, inspiré de (Spanos et al. 2011), compare les caractéristiques des différents outils de mapping détaillés précédemment.

Niveau d'automatisation : il peut être automatique, semi-automatique ou manuel.

Accessibilité des données : décrit la façon dont il est possible d'accéder au résultat du mapping. Les valeurs possibles sont ETL (Extraction, Transformation, Chargement), un langage de requête sémantique ou Web de données. Dans une approche ETL les données sont exportées de la base et stockées, soit dans un annuaire RDF, soit dans un fichier. En utilisant un langage de requête sémantique, notamment SPARQL, les données peuvent être interrogées dans la base de données d'origine sans avoir à les exporter. Dans ce cas il faut réécrire la requête sémantique en une requête SQL. La valeur Web de données signifie que les résultats sont publiés en suivant les principes du Web de données.

Langage de mapping : indique le langage dans lequel le mapping est exprimé

Langage de l'ontologie : définit le langage dans lequel l'ontologie concernée est exprimée. Suivant les approches il peut s'agir du langage de l'ontologie créée ou de l'ontologie utilisée.

Réutilisation du vocabulaire : ce paramètre permet de définir si une base de données relationnelle peut être mappée avec plus d'une ontologie. La réutilisation des termes issus de vocabulaires standards est un des éléments clé du Web Sémantique, il est donc important que les approches de mapping entre bases de données et ontologies l'autorise.

Disponibilité du logiciel : définit si l'approche est suivie de l'implémentation d'un logiciel permettant de vérifier les méthodes théoriques présentées.

GUI : définit si une interface graphique est présente avec le logiciel. Il s'agit d'un paramètre important qui permet à un concepteur de base de données de créer des mappings sans avoir de connaissances dans les standards du Web Sémantique.

But principal : permet de définir le but principal de chaque approche suivant les motivations présentées dans la Partie 4.4. Ce paramètre n'implique pas que l'approche ne puisse pas être utilisée pour une autre motivation.

Type d'approche : définit le type d'approche en accord avec la taxonomie réalisée dans la Figure 17.

nom de l'approche	niveau d'automatisation	accessibilité des données	langage de mapping	langage de l'ontologie	réutilisation du vocabulaire	logiciel disponible	GUI	but principal	type d'approche
D2R MAP	Automatique / manuel	ETL	basé sur du XML	RDF	oui	oui	non	intégration de données	Sans ingénierie inverse
D2RQ / D2R Serveur	automatique / manuel	SPARQL / Web des	langage D2RQ basé sur RDF	RDFS	oui	oui	non	accès aux données basé sur l'ontologie	Sans ingénierie inverse
MAPONTO	semi automatique	-	basé sur XML	OWL DL	non	oui	oui	définition du sens d'un schéma relationnel	Ontologie existante
MASTRO / ODBA	manuel	SPARQL	représentation spécifique à l'outil	DL-Lite /OWL	non	oui	oui	accès aux données basé sur l'ontologie	Ontologie existante
MOMIS	semi automatique	ETL / langage de requête OQL_{i3}	-	ODL_{i3}	non	oui	oui	intégration de données	Sans ingénierie inverse
Ontograte	Semi-automatique	Web-PDDL	-	Web-PDDL	non	oui	oui	intégration de données	Sans ingénierie inverse
R2O	semi automatique	ETL	Langage R2O	OWL DL	non	non	non	intégration de données	Ontologie existante
R2O + ODEMapster	manuel	ETL / langage de requête ODEMQL	langage R2O	OWL	oui	oui	non	accès aux données basé sur l'ontologie	Ontologie existante
RelationalOWL	automatique	ETL / SPARQL	-	OWL Full	non	oui	oui	échange de données	Ontologie dirigée structurellement
ROSEX	automatique	SPARQL	ontologie spécifique	OWL DL	non	non	non	accès aux données basé sur l'ontologie	structurellement + ingénierie inverse
Tirmizi et al	automatique	ETL	-	OWL DL	non	non	non	génération de données Web Sémantique	Ingénierie inverse
Virtuoso RDF Views	automatique / manuel	ETL / SPARQL / Web des Données	langage de méta-schéma Virtuoso	RDFS	oui	oui	oui	accès aux données basé sur l'ontologie	Sans ingénierie inverse

Tableau 1 : récapitulatif des caractéristiques principales des outils de mapping entre base de données et ontologies

4.7. La réécriture de requêtes SPARQL en requêtes SQL

L'intérêt des approches permettant un accès aux données basé sur les ontologies peut être comparé à l'intérêt des adaptateurs dans un système d'intégration. Les deux approches permettent de cacher les détails de la source en transformant les requêtes posées sur le schéma global en requêtes interrogeant la source de données d'origine. Dans le cas de requêtes posées sur une ontologie, cela signifie la transformation de requêtes sémantiques, souvent écrites en SPARQL, en requêtes basées sur l'algèbre relationnel ou en SQL. L'arrivée de stratégies de transformation de SPARQL en SQL efficaces (Cyga 2005; Elliott et al. 2009; Chebotko et al. 2009) a conduit les approches d'accès aux données, basées sur les ontologies, à être considérées comme une alternative aux annuaires RDF. Les bases de données, avec leurs stratégies d'optimisations robustes peuvent dépasser les performances d'exécution de requêtes des annuaires RDF notamment pour des gros jeux de données (Christian Bizer & Schultz 2009). Tant que cela sera le cas, l'accès dynamique à des bases de données relationnelles basées sur un mapping entre ontologies et bases de données restera une solution très performante. Le temps d'execution d'une requête SPARQL transformée en SQL est, dans le meilleur des cas, supérieure de 106% au temps d'execution d'une requête SQL (Christian Bizer & Schultz 2009). Cela signifie qu'il reste toujours la possibilité d'améliorer les performances de ces algorithmes de réécriture de requête.

4.8. Conclusion

Nous avons présenté dans cette partie les différentes approches existantes pour réaliser des mappings entre bases de données relationnelles et ontologies. Nous avons commencé par présenter l'utilisation de ces mappings pour des motivations variées comme l'intégration de données, l'annotation sémantique de pages Web dynamiques, la création massive de données sémantiques, l'aide à la création d'ontologies ou l'accès aux données basé sur une ontologie. Nous avons également présenté une taxonomie des différentes approches, taxonomie débouchant à six classes, chacune de ces classes n'étant pas spécifique à une motivation donnée.

Les approches d'accès aux données basé sur une ontologie ont été comparées à des approches de création d'adaptateur sémantique par leur capacité à faire le pont entre un schéma global, représenté par l'ontologie, et le schéma local d'une base de données relationnelle. Ces

approches sont complétées par des algorithmes de transformation de requête permettant de transformer une requête sémantique, exprimée sur une ontologie, en une requête relationnelle pouvant interroger la base de données. La littérature a mis en avant que ces approches d'accès dynamique aux bases de données offrent de meilleurs performances d'interrogation que les approches de type annuaire RDF.

Contexte et objectifs

5. Contexte et objectifs

5.1. Le Contexte scientifique

Ce projet de thèse s'intègre dans la continuité du large projet international du Generation Challenge Program (GCP) (Richard Bruskiewich et al. 2006) SP4 (Subprogramme 4, Bioinformatics and Crop Information Systems), dans lequel l'équipe l'Intégration de Données de l'UMR Amélioration Génétique et Adaptation des Plantes a eu une participation active.

Le projet GCP SP4, tout en se basant sur les initiatives déjà existantes (Gene Ontology (Michael Ashburner et al. 2000) , Plant Ontology (Jaiswal et al. 2005) , Chado (C. J. Mungall et al. 2007) , etc.), a développé de nouveaux standards complémentaires pour la description des données génétiques chez les plantes cultivées (Richard Bruskiewich et al. 2006). Ce projet a aussi mis en place une plateforme d'intégration des données à base de médiation, GCP Pantheon[8], qui utilise les ontologies et la plateforme de Web Services BioMoby[9]. Le médiateur intègre les adaptateurs à travers un schéma global, qui dans notre cas correspond à une ontologie de domaine. Les applications développées sont librement disponibles et ont fait l'objet de publications (Richard Bruskiewich, Martin Senger, G. Davenport, Ruiz, Rouard & al 2008; Wanchana et al. 2008).

Cependant ce projet a révélé un certain nombre de limitations, que nous présenterons dans ce chapitre. Ces limitations n'ont pas permis une adoption rapide de cette plateforme par tous les fournisseurs de données biologiques du projet GCP. Un travail de recherche méthodologique s'avérait nécessaire pour lever certains verrous, et cette réflexion a guidé et orienté ce travail de thèse.

5.1.1. Le Projet GCP

Le *Generation Challenge Programme* (GCP) (Richard Bruskiewich et al. 2006) est l'un des cinq *Challenge Programme* établis par le *Consultative Group on International Agricultural Research* (CGIAR). Le consortium du GCP est présent sur tous les continents (Figure 18). Il met en relation neuf centres dépendants du CGIAR, six *Advanced Research*

[8] http://pantheon.generationcp.org/
[9] http://biomoby.org/

Institutes (ARI), situés dans les pays du nord, et sept *National Agricultural Research System* (NARS), situés dans les pays du sud.

Le but du GCP est d'utiliser la diversité génétique des plantes, la génomique fonctionnelle et les avancées en biologie moléculaire pour développer des outils et des technologies qui permettraient aux sélectionneurs des pays en voie de développement de créer de meilleures variétés de plantes (e.g plus productives, plus tolérantes à la sécheresse ou à la salinité). Le programme s'intéresse à toutes les plantes étudiées par les centres du CGIAR que ce soit des plantes utilisées largement dans l'alimentation mondiale tels que le blé, le riz ou encore la bananier, et des plantes ayant une grande importance dans certains pays mais peu étudiées par la communauté scientifique comme c'est le cas du millet.

Figure 18 : Répartition géographique des organismes membres du programme GCP

5.1.1.1. La plateforme GCP Pantheon

Le programme GCP aborde des thématiques diverses à travers ses sous programmes qui créeent des données hétérogènes. Ces données sont réparties physiquement dans les sources de données de chaque collaborateur. Une plateforme d'intégration de données, nommée GCP Pantheon, a été développée afin de permettre à des clients d'interroger de manière transparente tout type de données générées dans le cadre du programme GCP. Cette plateforme combine une approche de médiation LAV (*Local As View*) et une approche

sémantique, permettant aux partenaires de gérer leurs données localement puis de les rendre accessibles facilement en accord avec un modèle conçu spécifiquement pour le projet, *GCP Domain Model* (Richard Bruskiewich et al. 2006). Ce modèle peut être utilisé quel que soit le langage d'implémentation de l'élément que l'on souhaite intégrer à la plateforme. L'architecture de Pantheon est présentée dans la Figure 19 issue de (Richard Bruskiewich, Martin Senger, G. Davenport, Ruiz, Rouard & al 2008). Cette architecture décrit une sous-couche logicielle, basée sur le modèle GCP ainsi qu'une API Java Pantheon, utilisée pour accéder de manière transparente à des sources de données ainsi qu'à des outils de traitement de données. Le modèle GCP a également été utilisé pour implémenter un réseau GCP spécifique à Pantheon, basé sur des protocoles d'échanges de données tels que BioMoby (M. Wilkinson et al. 2005), SoapLab (Martin Senger et al. 2009), SSWAP (D. D. G. Gessler, G. S. Schiltz, et al. 2009) et Tapir (De Giovanni et al. 2010). Mais BioMoby, par la facilité de son utilisation et le dynamisme de sa communauté, s'est rapidement imposé dans le développement des SW du projet.

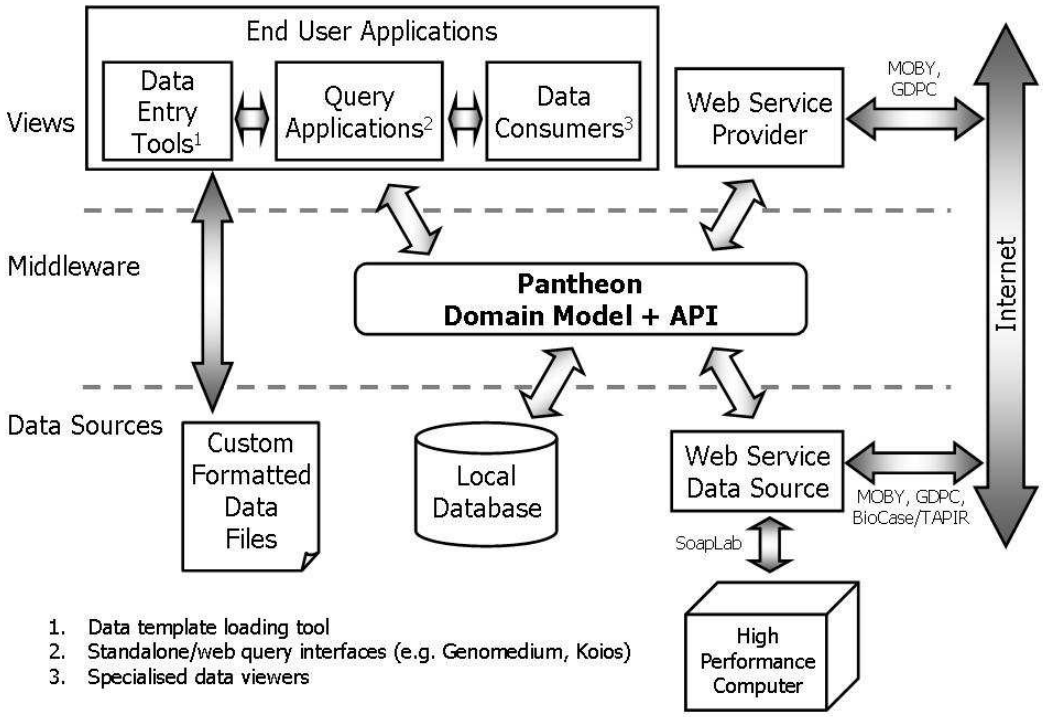


Figure 19 : Architecture de la plateforme Pantheon (Richard Bruskiewich, Martin Senger, G. Davenport, Ruiz, Rouard & al 2008)

5.1.2. BioMoby

BioMoby est un une plateforme d'interopérabilité, *open source,* basé sur les SW et les ontologies (*cf* chapitre 3). Le projet était initialement composé de deux branches appelées *BioMoby Service* et *Semantic BioMoby*. *Semantic BioMoby* fut par la suite rebaptisé SSWAP.

Dans cette partie nous parlerons uniquement de *BioMoby Service*, utilisée par GCP Pantheon, et qui offre une architecture pour la découverte et la distribution de données biologiques à l'aide de SW. Les données et les SW sont décentralisés mais la disponibilité de ces ressources et les instructions permettant d'interagir avec eux sont enregistrées dans un annuaire central nommé *Moby Central*.

5.1.2.1.Les Services Web Sémantiques

La sémantique des SW BioMoby est basée sur 3 ontologies :

- **Ontologie de type de données** : définit les types de données utilisés en entrée et sortie des SW. Cette ontologie peut être représentée sous la forme d'un graphe ou chaque nœud représente une classe et chaque arrête représente une relation entre deux classes. Cette ontologie comporte trois types de propriétés :

  La première est appelée « IS A » et définit une relation d'héritage entre deux classes. Toutes les propriétés de la classe parent sont transmises à la classe fille. Les deux autres propriétés sont des conteneurs qui diffèrent uniquement par leur cardinalité. Ainsi la propriété « HAS A » définie un conteneur de cardinalité 1 alors que la propriété « HAS » définie un conteneur de cardinalité 1 ou plus.

  La Figure 20 représente quatre classes de l'ontologie de types de données. La classe *GCP_SimpleIdentifier* contient un nom, la classe *GCP_Identifier* contient une description, une version, et un nom hérité de la classe *GCP_SimpleIdentifier*. La classe *GCP_Namespace* contient un ou plusieurs composants ainsi que toutes les propriétés héritées de *GCP_Identifier*. Les composants possèdent un nom. Par exemple *GCP_SimpleIdentifier* possède un composant HASA de type *String*, et ce composant a pour nom *name*. Cela signifie que *GCP_SimpleIdentifier* est composé d'une chaîne de caractère correspondant à un nom. Le nom d'un composant est exprimé dans un texte libre, et il ne possède donc aucune sémantique.
- **Ontologie de type de services** : définit le type d'action réalisée par le SW. Il est ainsi possible de définir qu'un SW utilise un outil d'alignement en l'annotant avec la classe

Alignment ou qu'un SW renvoie une donnée contenue dans une base de données biologique en l'annotant avec la classe *Retrieval*.

- **Ontologie d'espace de noms** : définit de manière unique une ressource biologique, comme par exemple GO_ID permettant de représenter tout type d'entité issue de la bioontologie *Gene Ontology*.

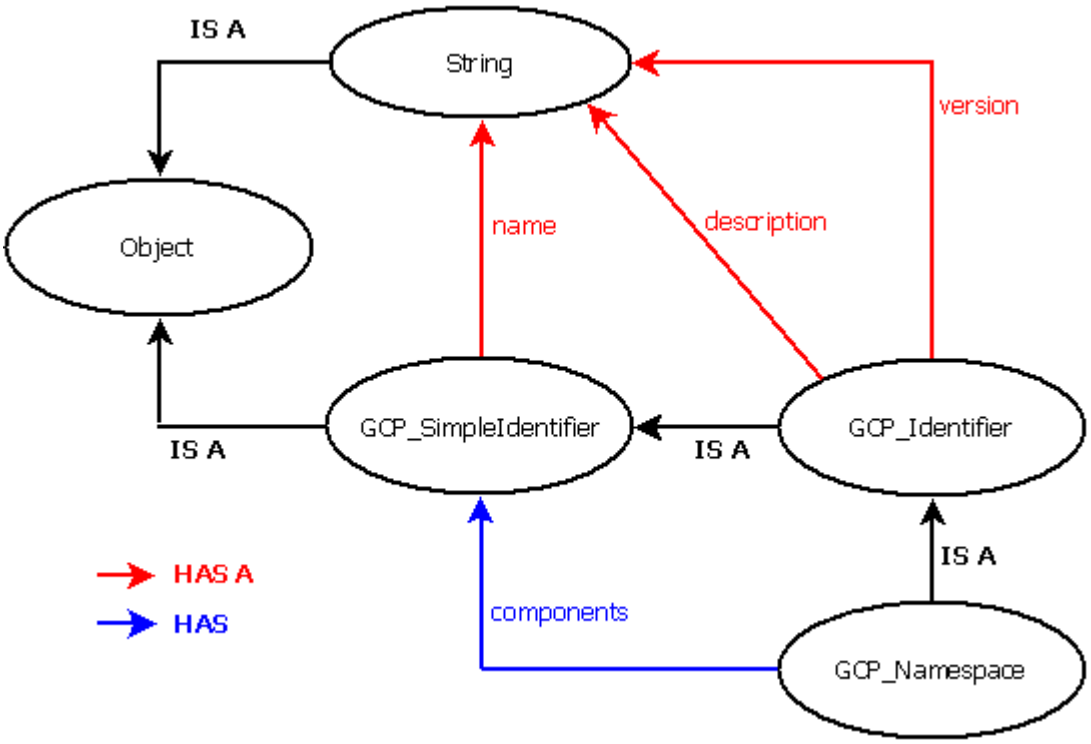


Figure 20 : représentation de types de données BioMoby

Ces ontologies permettent aux SWS BioMoby d'être découverts plus facilement (M. Wilkinson et al. 2005), d'être utilisés pour faire de la composition semi-automatique de SW (DiBernardo et al. 2008; Kawas et al. 2006), et de permettre l'interopérabilité entre bases de données biologiques (M. Wilkinson et al. 2003).

5.1.2.2.BioMoby DashBoard

BioMoby Dashboard est un outil développé en Java, offrant une interface graphique conviviale pour l'API BioMoby et permettant d'automatiser certaines étapes de la création de SWS BioMoby. Cet outil est composé de cinq onglets graphiques différents permettant chacun de réaliser une action différente. Les actions réalisables dans chacun des onglets sont présentées ci dessous :

- Registry Browser : fenêtre graphique permettant de visualiser toutes les entités enregistrées dans un annuaire Moby Central, aussi bien les classes des trois ontologies BioMoby que les SWS enregistrés. Cette fenêtre permet également de détecter des SWS en fonction de leur nom, de leurs concepts d'entrée ou de sortie, de leur espace de nom ou du type d'action qu'ils réalisent.

- Registration : fenêtre graphique composée elle-même de 4 onglets permettant respectivement d'enregistrer un nouveau type de données, un SWS, un espace de nom ou un type d'action.

- Simple Client : permet de sélectionner un SWS et de le tester en rentrant une valeur dans chaque champ correspondant à une entrée. Cette fenêtre est principalement utilisée par les développeurs afin de tester la fonctionnalité d'un service nouvellement enregistré.

- Perl MoSeS Generator : fenêtre graphique permettant de créer des SWS BioMoby implémentés en Perl et non pas en Java.

Pour pouvoir enregistrer un SW BioMoby il faut dans un premier temps enregistrer les classes que l'on souhaite utiliser, ou détecter les classes existantes pour chacune des trois ontologies. L'ontologie d'espace de nom nécessite l'enregistrement d'une classe pour la totalité des SW qu'un fournisseur de données souhaite enregistrer. L'ontologie de type d'action est suffisamment complète pour qu'une classe existante corresponde au type d'action à réaliser.

5.1.2.3.L'enregistrement des types de données

L'étape la plus longue à réaliser avant d'enregistrer un SWS BioMoby est l'enregistrement des classes de l'ontologie de type de données. Chaque classe de cette ontologie doit être enregistrée manuellement. La Figure 21 montre la fenêtre d'enregistrement d'un type de données. Pour en enregistrer un, il faut renseigner différents champs présents dans la partie centrale de la fenêtre, mais il faut surtout créer manuellement chaque propriété reliant notre classe à une classe déjà existante. Pour cela il faut parcourir l'ontologie de type de données (partie droite de la fenêtre), trouver la classe qui nous intéresse, faire un clic droit dessus puis sélectionner le type de propriété que l'on souhaite créer vers cette classe, comme on peut le voir dans l'encadré dans la partie droite de la Figure 21. Il faut répéter ces étapes pour chaque type de données que l'on souhaite associer au type de donnée en cours de création.

5.1.2.4.La création d'un Service Web

Dashboard utilise MoSeS (*Moby Services Support)* qui permet de créer des SWS de manière semi-automatisée. La création d'un SWS BioMoby se déroule en cinq étapes :

- Enregistrement du SW : il faut remplir tous les champs de la fenêtre présentée dans la Figure 22. En plus de ces champs, il faut également sélectionner la classe ontologique correspondant à l'espace de nom de fournisseur de données, et la classe ontologique correspondant au type d'action réalisée par SWS. Ces deux informations sont renseignées dans les parties représentées par des encadrés rouges dans la Figure 22. Il faut également sélectionner les classes ontologiques correspondant aux types de données d'entrée et de sortie du SW. Ceci est réalisé dans les parties représentées par des encadrés bleus dans la Figure 22. La validation de ces informations engendre l'enregistrement d'un SW dans l'annuaire Moby Central.

- Génération des classes Java : un fichier .jar contenant les objets Java nécessaire à l'utilisation des types de données sélectionnés en entrée et sortie est créé automatiquement.

- Génération du squelette de SW : un fichier .java contenant un squelette d'implémentation de la classe Java à implémenter est créé automatiquement.

- Implémentation du SW : cette étape consiste à implémenter manuellement une classe Java qui réalise le traitement des données souhaité. Cette classe étend le squelette créé automatiquement.

- Déploiement du SW : Cette étape consiste à déployer le SW sur un serveur Tomcat.

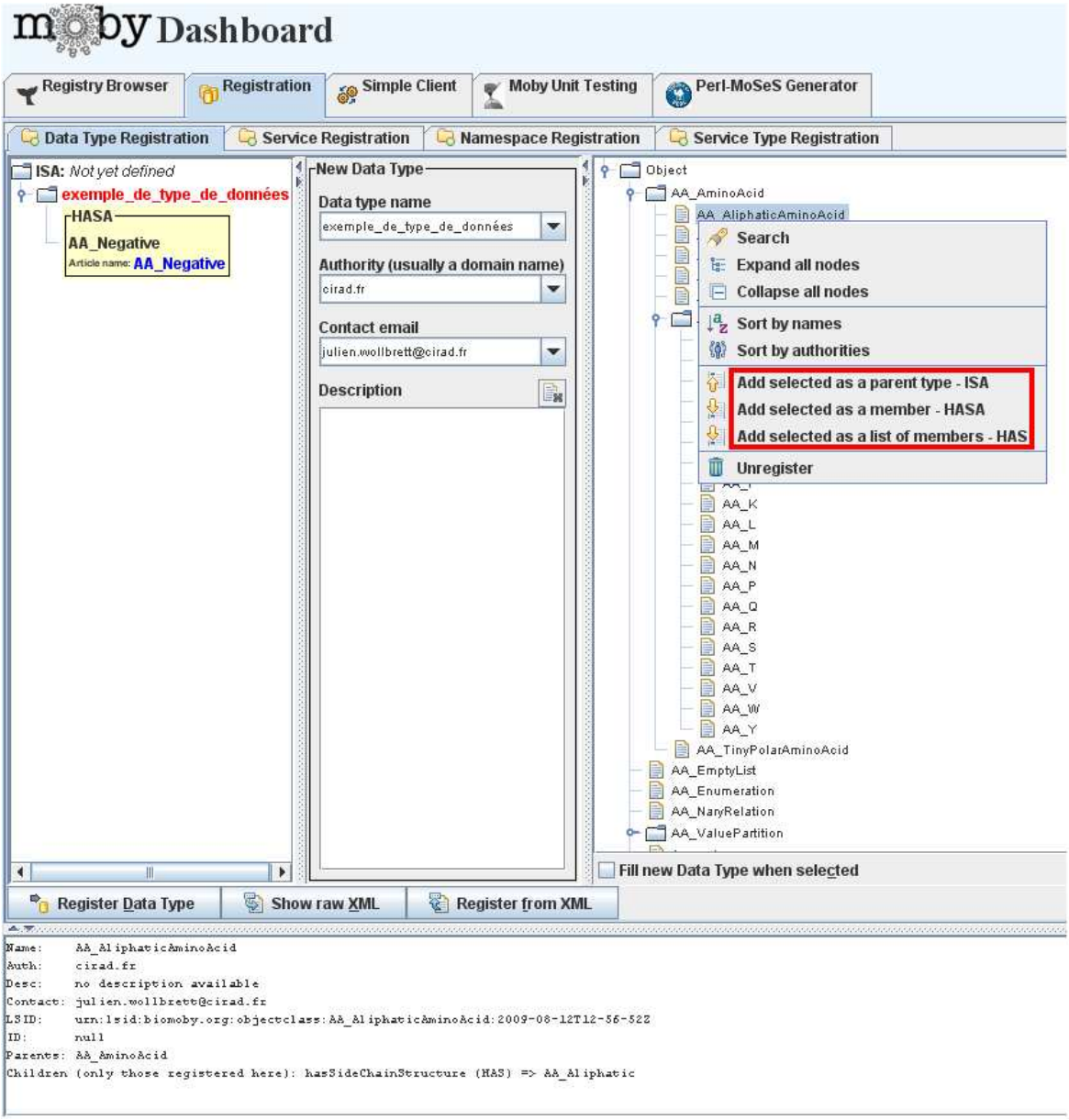


Figure 21 : Fenêtre permettant l'enregistrement d'un type de données sous BioMoby Dashboard

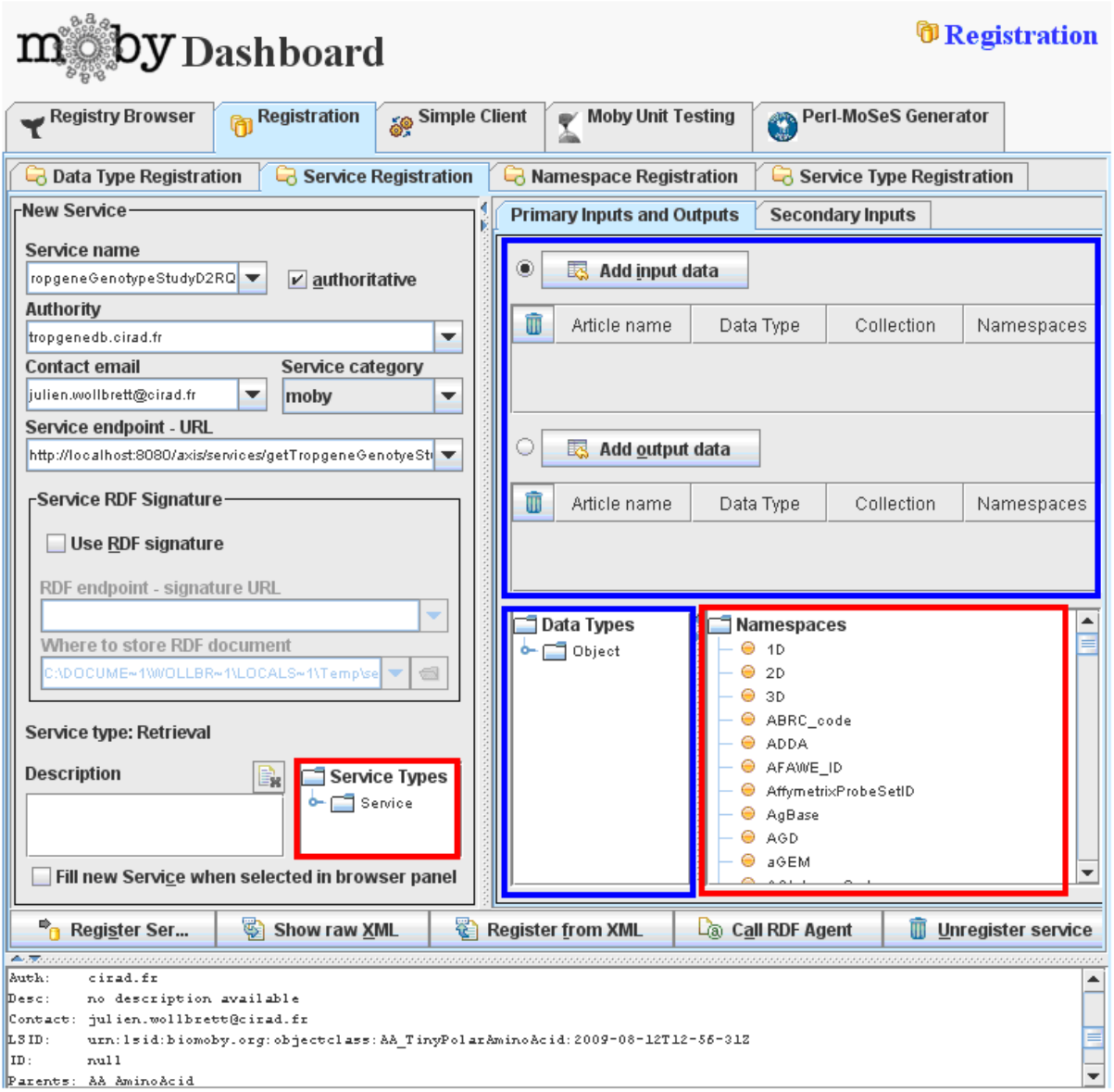


Figure 22 : Fenêtre d'enregistrement d'un Service Web sous BioMoby DashBoard

5.1.3. Les avantages de GCP Pantheon

GCP Pantheon est une plateforme d'intégration de données biologiques et d'outils d'analyses de données. Cette plateforme est basée sur une approche de médiation permettant à des chercheurs ou des agronomes d'interroger les sources de données intégrées virtuellement. GCP Pantheon est facilement utilisable par les consommateurs de données et permet l'ajout de nouvelles sources de données ou de nouveaux outils d'analyses. Cela lui permet d'augmenter l'interopérabilité à travers les données des partenaires du projet GCP et d'étendre cette capacité plus tard à travers toute la communauté de chercheurs agronomes. Le schéma global *GCP Domain Model* a une couverture large des types de données possibles du

domaine, allant des analyses en laboratoire jusqu'à la manipulation de germplasmes, tout cela couplé à un géoréférencement. L'intégration couplée de sources de données et d'outils d'analyses permet aux agronomes de répondre facilement à des questions complexes en disposant d'un grand pool de données. La plateforme GCP Pantheon permet par exemple de répondre aux questions suivantes :

- renvoyer la liste de toutes les cartes génétiques incluant un locus à effets quantitatifs (QTL) pour un caractère phénotypique donné

- renvoyer toutes les données passeport ainsi que les informations sur le génotype et le phénotype pour un germplasme donné.

5.1.4. Les limites de la plateforme GCP Pantheon

La plateforme GCP Pantheon a atteint son objectif initial qui était de fournir une plateforme d'intégration de données et d'outils d'analyses pour l'analyse de la diversité génétique de plantes. La plateforme permet de répondre à des questions complexes et permet l'ajout de nouvelles sources de données.

Cependant la connexion de nouvelles sources de données à la plateforme GCP Pantheon demande un temps de développement non négligeable pour le fournisseur de données. L'ajout de nouvelles sources implique en effet la maîtrise de plusieurs composants de la plateforme: une connaissance et une compréhension du GCP domain model, la capacité à développer des SW BioMoby, et une connaissance de l'API java Pantheon. Plusieurs étapes sont non automatisées et nécessitent des compétences différentes : le mapping entre la base de données à connecter et les concepts du *GCP domain model* concernent surtout le gestionnaire de la base de données et/ou les biologistes utilisateurs, alors que le développement des WS et la connection à Pantheon nécessite des compétences importantes en développement informatique.

En outre, le développement d'adaptateurs est spécifique à la plateforme GCP Pantheon. Les adaptateurs ne pourront pas être réutilisés dans une autre plateforme basée sur un schéma global différent du *GCP domain model*.

Cette difficulté à rajouter rapidement de nouvelles sources de données ainsi que le manque de généricité des adaptateurs empêchent l'adhésion des partenaires et de tous les chercheurs de la communauté. Or le succès d'une plateforme d'intégration dépend du nombre de sources de données effectivement connectées.

5.2. Les objectifs de la thèse

5.2.1. Objectif général

Ce travail de thèse se place donc dans le cadre d'approches d'intégration de données par médiation qui permettent d'intégrer virtuellement des bases de données. Dans ce cadre, les données sont accessibles de manière transparente aux utilisateurs sans pour autant être exportées de leur source d'origine. Cela évite toute gestion de mise à jour des données car les données restent dans les sources. Dans le contexte de sources de données biologiques évoluant très rapidement, cet avantage est non négligeable. Les systèmes de médiation et plus précisément les systèmes de médiation LAV, facilitent l'ajout de nouvelles sources de données. Cet ajout de nouvelles sources se fait par la création d'adaptateurs permettant de transformer une requête exprimée dans le schéma global du médiateur en une requête compréhensible par le schéma local de la source de données.

Le dévelopement des adaptateurs qui est, en général, réalisée par les fournisseurs de données doit être automatisé au maximum. En outre, il est souhaitable que les adaptateurs soient basés sur des standards de manière à pouvoir être réutilisés dans différentes plateformes d'intégration. Nous cherchons donc à mettre en place une méthode de création d'adaptateurs à base de SWS, couplant les technologies et les avantages des Services Web et du Web Sémantique.

La majorité des données biologiques publiques disponibles sur le Web sont issues de bases de données relationnelles. Ce travail se focalise donc sur la création d'adaptateurs sémantiques permettant l'intégration de sources de données de type base de données relationnelles.

5.2.2. L'approche basée sur BioMoby

Dans un premier temps, notre projet s'est porté sur l'amélioration de l'approche mise en place pour la création d'adaptateurs sémantiques dans la plateforme GCP Pantheon, à l'aide de SWS BioMoby. La contribution à cette approche est présentée dans le **chapitre 6**. La première partie décrit l'architecture globale de la plateforme conçue pour répondre à nos objectifs. Notre approche est organisée en trois étapes qui consistent respectivement à enregistrer automatiquement une bioontologie de domaine dans l'ontologie de types de données BioMoby, à créer des correspondances entre les bases de données relationnelles existantes et les bioontologies de domaine, et enfin à utiliser les deux étapes précédentes pour

créer automatiquement des SWS à l'aide de BioMoby. Nous présentons également la solution proposée, sous la forme d'un plugin Protégé nommé BioMoby Converter, qui répond à la première étape. Ce chapitre conclut en présentant les limites de BioMoby qui ont justifé l'abandon de son utilisation pour la suite du projet.

5.2.3. L'approche basée sur les standards du Web Sémantique

Ayant abandonné l'utilisation de BioMoby, nous nous sommes concentrés sur la création d'une plateforme générique par médiation, automatisant la création d'adaptateurs sémantiques.

Le Web Sémantique fournit les technologies nécessaires à l'intégration de sources de données biologiques et à la représentation de connaissances. L'utilisation de ces technologies standards doit nous permettre de rendre nos données interopérables avec la grande quantité de données biologiques disponibles sur le Web des Données.

Les Services Web fournissent une structure permettant l'interopérabilité d'applications à travers le Web en se basant sur des protocoles standards. Ils permettent d'encapsuler tout type d'application dans cette structure, la rendant ainsi indépendante de toute plateforme et facilement réutilisable. Dans notre approche, nous souhaitons créer des adaptateurs de type Services Web Sémantique. Cette composante sémantique supplémentaire permettra, en annotant leurs entrées et sorties, de faciliter leurs détections, leurs compositions, et la création de flux de travaux.

Les adaptateurs d'une plateforme de médiation permettent de faire le pont entre le schéma global représenté dans notre cas par des bioontologies et le schéma local représenté dans notre cas par le schéma des bases de données relationnelles à intégrer. La création automatisée de nos adaptateurs est donc dépendante de la mise en correspondance entre une ou plusieurs ontologies et des schémas de bases de données relationnelles. Cette mise en correspondances doit permettre de cacher les détails de la source en transformant les requêtes posées sur le schéma global en requêtes interrogeant la source de données d'origine. Ce type de correspondances est réalisé par les approches d'accès aux données basé sur une ontologie. L'approche que nous allons utiliser permettra de créer automatiquement une ontologie locale sous la forme d'une vue du schéma de la base de données relationnelle. Cette vue devra être exprimée dans un langage basé sur les standards du Web Sémantique et être annotée sémantiquement à l'aide de concepts ontologiques.

Le détail de la plateforme mise en place ainsi que la description de l'approche utilisée pour créer des Services Web Sémantiques sont présentés dans le **chapitre 7**.

Un adaptateur doit intégrer une requête permettant d'interroger la source de données d'origine. Cela est rendu possible par l'encapsulation d'une requête, exprimée dans un langage compréhensible par la base de données relationnelle à interroger, dans un SWS annoté à l'aide de bioontologies. De manière à automatiser au maximum la création de ces adaptateurs nous présentons dans le **chapitre 8** une approche originale de création automatique de ces requêtes.

Contributions

6. Intégration automatique de bioontologies de domaine dans BioMoby

6.1. Introduction

GCP Pantheon est une plateforme d'intégration de données, à base de médiation et ayant une composante sémantique. Mon travail de thèse consistait initialement à la création automatisée d'adaptateurs spécifiques à Pantheon. Ces adaptateurs utilisent BioMoby, qui permet la création de Services Web Sémantiques (SWS) en automatisant certaines étapes de leur implémentation. La sémantique des Services Web (SW) BioMoby est apportée par les différentes ontologies spécifiques à BioMoby, qui typent les entrées et sorties des SW. Dans BioMoby, chaque nouveau concept ontologique doit être ajouté manuellement, ce qui limite la possibilité d'enregistrer des bioontologies de domaine déjà existantes. Afin de dépasser cette limitation nous nous sommes interessés à l'automatisation de l'enregistrement de concepts bioontologiques déjà existants, dans une ontologie de type BioMoby.

6.2. L'approche générale

Notre approche générale pour la création de SWS BioMoby pour de nouvelles bases de données relationnelles peut être divisée en trois parties (Figure 23). La première étape consiste à enregistrer des bioontologies de domaine dans l'ontologie de type de données BioMoby. Une bioontologie ne doit être enregistrée qu'une seule fois pour ensuite pouvoir être utilisée pour créer des SWS. La deuxième étape consiste à mettre en correspondance le schéma relationnel d'une base de données relationnelle et des concepts issus de bioontologies. Cette étape n'est à réalisée qu'une seule fois pour permettre ensuite la création de plusieurs SWS. Les étapes 1 et 2 sont indépendantes mais sont des étapes préliminaires indispensables à l'étape de création de SWS.

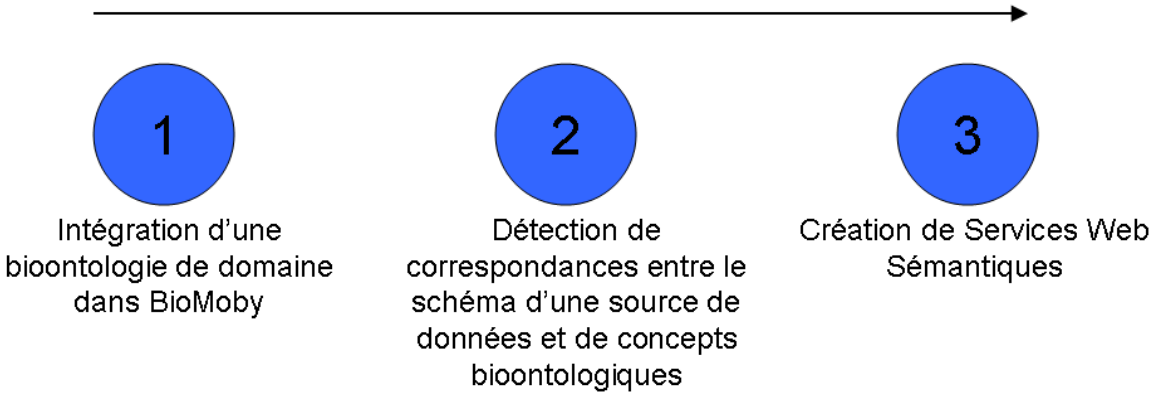


Figure 23 : Présentation des 3 étapes d'intégration

L'architecture générale d'intégration de nouvelles sources de données est proposée et présentée dans la

Figure 24. Les trois étapes y sont représentées d'une couleur différente, et une quatrième étape, correspondant à la détection et l'interrogation d'un SW, a été ajoutée (rouge).

Dans la première étape (vert) tous les concepts d'une ontologie OWL sont enregistrés dans l'ontologie de types de données BioMoby. En raison de la différence d'expressivité entre ces deux types d'ontologies, et afin de conserver l'expressivité initiale des ontologies OWL, une correspondance entre les concepts issus de la bioontologie et ceux issus de l'ontologie de type de données BioMoby est conservée dans une base de données.

Dans la deuxième étape (orange), l'ontologie OWL est mise en correspondance avec les éléments d'un schéma relationnel de base de données. Ces correspondances sont également stockées dans une base de données.

Pour que la troisième étape (gris) puisse être réalisée et que des SWS BioMoby puissent être créés, il faut obligatoirement que les concepts ontologiques mis en correspondance avec le schéma relationnel de la base de données soient enregistrés dans l'ontologie de types de données BioMoby. Si c'est le cas, des SWS peuvent être créés en sélectionnant ses concepts ontologiques d'entrée et de sortie. Ces SWS pourront ainsi être détectés et interrogés par un client directement dans l'annuaire Central BioMoby (rouge).

Cette architecture possède trois étapes manuelles rendant l'intégration d'une base de données relationnelle consommatrice de temps. Ces étapes sont l'enregistrement d'une bioontologie de domaine dans BioMoby, la détection de correspondances entre une ontologie

et un schéma relationnel, et la l'implementation des SWS. L'objectif est d'automatiser au maximum ces trois étapes.

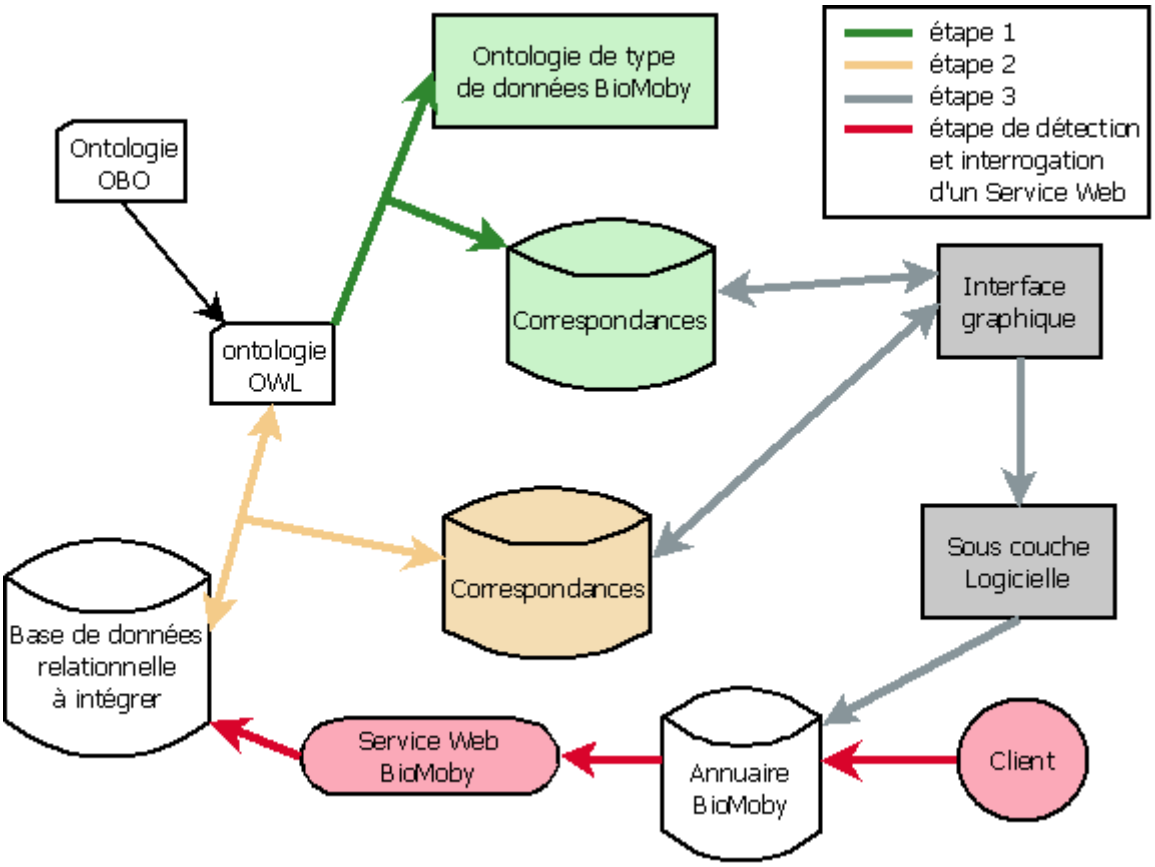


Figure 24 : Architecture d'intégration proposée

6.3. L'intégration automatique de bioontologies de domaine dans BioMoby

Comme présenté précédemment, l'ajout de type de données dans l'ontologie de types de données BioMoby doit être réalisé, un type de données à la fois. Dans notre approche, nous souhaitons utiliser des bioontologies déjà existantes comme par exemple *Gene Ontology* (GO) (Michael Ashburner et al. 2000) dont le but est de standardiser la description de gènes et de produits de gènes, ou *Sequence Ontology* (K Eilbeck et al. 2005) dont le but est de standardiser la description de séquences biologiques ainsi que leurs attributs. Or ces bioontologies possèdent un grand nombre de classes, par exemple GO en possède plus de 35000 classes, ce qui rend impossible son enregistrement classe par classe.

Il est également envisageable de vouloir enregistrer des types de données n'étant pas issus de bioontologies existantes. Le développement d'une ontologie est une tâche longue et sujette aux erreurs (López-garcía et al. 2009). Il existe des outils spécialisés dans le

développement et l'édition d'ontologies, permettant par exemple de la représenter sous forme de graphe ou de vérifier sa cohérence à l'aide de raisonneurs, facilitant ainsi grandement sa manipulation. Il serait donc intéressant de créer un outil permettant de créer, modifier et visualiser une ontologie avant de l'enregistrer dans BioMoby.

Dans cette partie du mémoire nous présenterons uniquement le travail réalisé pour automatiser l'enregistrement d'une bioontologie de domaine dans l'ontologie de types de données BioMoby. Pour cela, nous avons développé BioMoby Converter (Wollbrett et al. 2009), un plugin pour le logiciel de création et d'édition d'ontologie Protégé[10].

6.4. La méthodologie mise en place

6.4.1. Le détail des besoins

Afin de déterminer précisément la méthodologie à mettre en place il convient dans un premier temps de définir les besoins auquel BioMoby Converter doit pouvoir répondre.

Prise en compte d'ontologies OWL et OBO : les bioontologies disponibles utilisent deux types de langage. Le langage OWL (Sean Bechhofer et al. 2004) issu de la communauté du Web Sémantique et le langage OBO (Chris Mungall & Ireland 2010) issu de la communauté bioinformatique. Il est donc important que BioMoby Converter puisse enregistrer des ontologies utilisant ces deux langages.

Création d'une ontologie : BioMoby Converter doit pouvoir permettre l'enregistrement d'ontologies déjà existantes mais également de permettre la création de nouvelles ontologies. Il est donc important de faciliter le travail des développeurs d'ontologie en leur offrant une interface performante et conviviale.

Visualisation d'une ontologie : une bioontologie pouvant contenir un grand nombre de classes, il est important que BioMoby Converter permette de représenter l'ontologie à intégrer sous forme de graphe.

Enregistrement : comme nous le précisions précédemment, l'objectif principal de notre outil sera de permettre l'enregistrement automatique d'une ontologie dans BioMoby.

Enregistrement d'une portion d'ontologie : une bioontologie peut contenir plusieurs milliers de classes, celles-ci ne sont pas forcément toutes spécifiques au domaine des sources à intégrer. Il faut donc pouvoir sélectionner la portion d'ontologie que l'on souhaite enregistrer.

[10] http://protege.stanford.edu/

Suppression d'une portion de l'ontologie de types de données : il est envisageable qu'un bioinformaticien ait enregistré une portion d'ontologie dont il n'a pas l'utilité. Il faut donc lui permettre de pouvoir supprimer des classes de l'ontologie de types de données BioMoby.

Export de l'ontologie de types de données au format OWL : BioMoby permet de créer son propre annuaire Central référençant uniquement les SW que l'on crée. Au départ, ce nouvel annuaire aura une ontologie de type de données qui sera vide. Une ontologie de type de données exportée à l'aide de BioMoby Converter pourra ensuite être enregistrée automatiquement dans un autre annuaire BioMoby. Cette option permettra également de vérifier la cohérence de l'ontologie OWL créée.

6.4.2. Détermination du Framework de développement

Des outils existants répondent déjà à certains besoins de BioMoby Converter. Des logiciels d'édition d'ontologies permettent de vérifier la cohérence d'une ontologie, facilitant ainsi leur création, et intègrent également des outils de visualisation performants. Protégé est le logiciel d'édition d'ontologie le plus utilisé. Sa popularité repose notamment sur son code Java open source et sur la présence d'une API facilitant la création de nouveaux plugins. Plusieurs dizaines de plugins sont actuellement disponibles[11] sur des sujets englobant l'inférence, le raisonnement, le Web Sémantique, la visualisation, le traitement de langage naturel ou encore la bioinformatique. Il existe par exemple 23 plugins de visualisation d'ontologie, et un plugin, nommé *obo-converter*, permettant de transformer automatiquement une ontologie OBO en une ontologie OWL. L'utilisation de Protégé nous permet de répondre à trois de nos besoins en donnant accès à un outil comportant déjà les fonctionnalités d'édition et de visualisation d'ontologie tout en permettant d'utiliser des ontologies au format OBO et OWL. Nous avons donc décidé de développé BioMoby Converter sous la forme d'un plugin de Protégé.

6.4.3. La conception de BioMoby Converter

L'outil que nous souhaitons développer doit permettre de réaliser l'étape d'enregistrement d'une ontologie dans BioMoby sans avoir à utiliser l'interface graphique BioMoby Dashboard. La Figure 25 présente les différents cas d'utilisations auxquels un utilisateur doit pouvoir répondre. L'utilisation principale de BioMoby Converter sera d'enregistrer ou de supprimer une ontologie dans BioMoby ou encore d'exporter l'ontologie

[11] http://protegewiki.stanford.edu/wiki/Protege_Plugin_Library

de types de données BioMoby vers une ontologie OWL. Pour cela un utilisateur devra choisir une action parmi celles proposées et remplir les champs nécessaires. La validation d'une action permettra de la réaliser automatiquement.

Dashboard permet une utilisation plus réactive en autorisant l'importation, dans un cache local, de données contenues dans un annuaire Central BioMoby. Afin d'accélérer l'utilisation de BioMoby Converter, un utilisateur pourra sélectionner un dossier de cache. Cela lui permettra de faire correspondre ce dossier avec celui de *Dashboard*.

BioMoby autorise la création de son propre annuaire Central. Chaque annuaire créé s'identifie par un point d'entrée et un espace de nom, tous deux correspondant à des URLs. Un utilisateur doit pouvoir facilement ajouter les informations vers un annuaire BioMoby existant et ainsi permettre à BioMoby Converter de s'y connecter.

Dashboard permet de visualiser l'arborescence de tous les types de données enregistrés. Une telle visualisation est nécessaire à notre outil afin de visualiser les types de données présents dans un annuaire Central. L'utilisateur doit avoir accès aux derniers types de données enregistrés dans un annuaire, il faut donc lui permettre de pouvoir mettre à jour cette arborescence en temps réel.

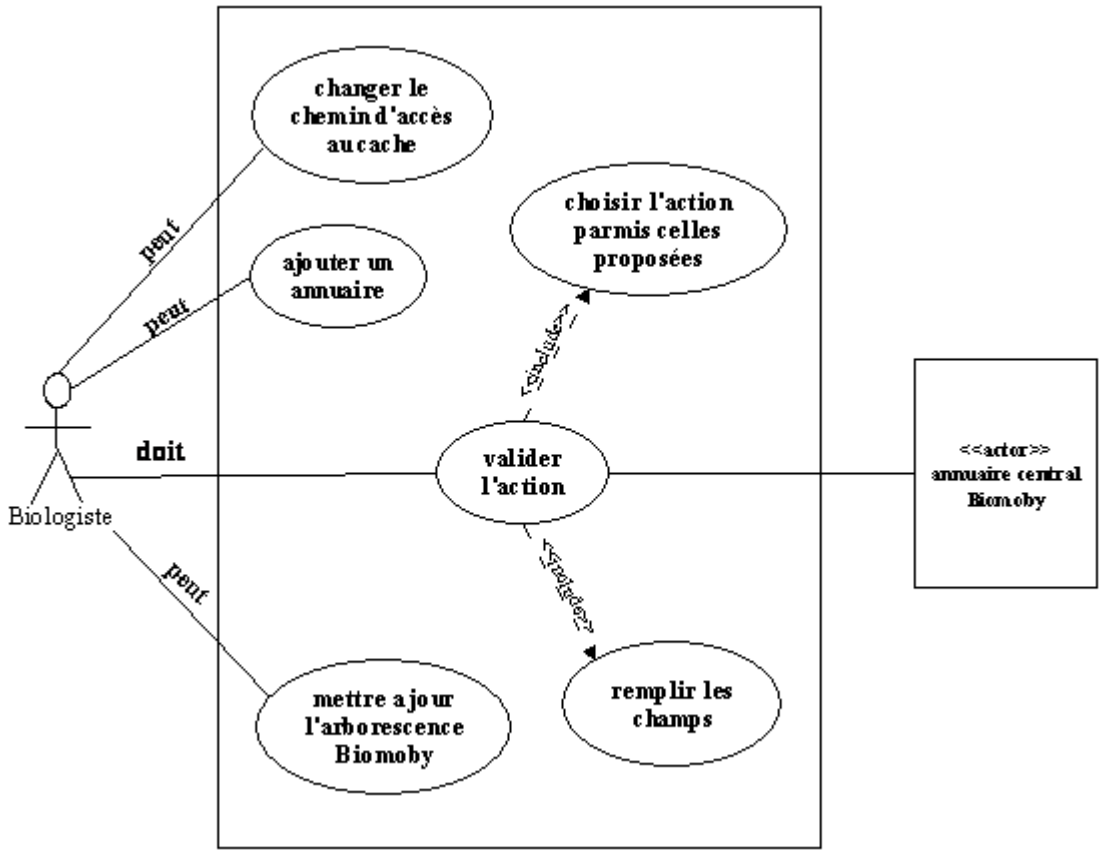


Figure 25 : Diagramme de cas d'utilisation de BioMoby Converter

Le flux de travaux que doit réaliser notre plugin est présenté dans le diagramme d'activité de la Figure 26. Lorsque le plugin est chargé dans Protégé, la première étape consiste à sélectionner le type d'action à réaliser. Chaque action nécessitera de récupérer des informations différentes qui pourront être saisies par l'utilisateur sous la forme de champs à remplir.

Les ontologies OWL et les ontologies OBO transformées en OWL ont des différences au niveau de leur formalisme. Dans le cas de **l'enregistrement** d'une ontologie, le programme doit donc déterminer dans un premier temps le formalisme de l'ontologie afin de la parser correctement. Dans un second temps le programme doit récupérer les informations correspondant à toutes les classes ontologiques à enregistrer. Cette étape correspondra à la sélection de toutes les classes dans le cas de l'enregistrement d'une **ontologie complète**.

Il est également possible d'enregistrer une portion d'ontologie. Une ontologie peut être représentée sous la forme d'un graphe orienté ou chaque nœud représente une classe. L'enregistrement d'une **portion d'ontologie** revient donc à enregistrer un sous graphe. Pour cela, il faut sélectionner le nœud parent de tous les nœuds qu'on souhaite enregistrer. Le programme se chargera ensuite de sélectionner toutes les classes correspondant aux nœuds de ce sous graphe. Un nouveau type de données BioMoby ne peut être enregistré que si tous les types de données pour lesquels il possède des propriétés (ISA, HAS, HASA) sont déjà enregistrés. Cela nécessite de trier les classes afin de les enregistrer.

Lorsque l'action souhaitée est l'**export** de l'ontologie de types de données BioMoby vers une ontologie OWL, la première étape à réaliser est la récupération de tous les types de données. Il faut ensuite créer les classes OWL en prenant compte des propriétés des types de données. Lorsque toutes les classes OWL ont été créées, elles sont écrites dans un fichier OWL.

Lorsque l'action souhaitée est la **suppression** de types de données BioMoby, la première étape consiste à sélectionner le sous graphe ou chaque nœud correspondra à un type de données à supprimer. Il faut donc sélectionner le nœud parent de tous les nœuds à supprimer pour que le programme les sélectionne. Il n'est pas possible de supprimer un type de données BioMoby si un autre type de données y est relié par une propriété. Il faut donc trier les types de données à supprimer avant d'appliquer la suppression.

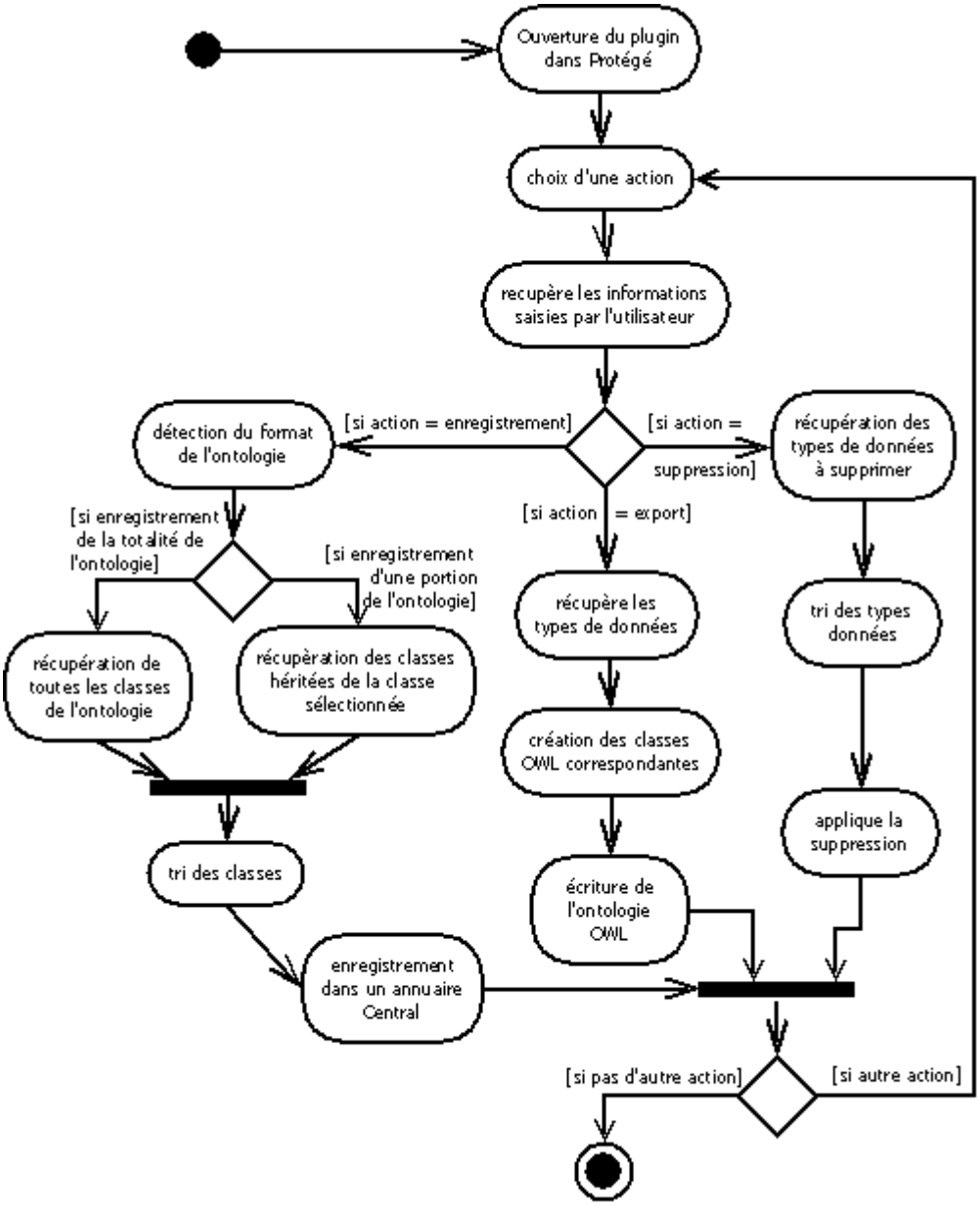


Figure 26 : Diagramme d'activité de BioMoby Converter

6.5. L'implémentation du plugin

6.5.1. Le parcours d'une ontologie OWL

Les fichiers OWL peuvent être parsés à l'aide d'API Java comme Jena ou OWLAPI. Jena, qui est la librairie utilisée dans Protégé 3.4, permet d'utiliser des requêtes SPARQL mais n'est pas compatible OWL 2.0 alors qu'OWLAPI, qui est utilisé dans Protégé 4, est compatible OWL 2.0 mais ne gère pas les requêtes SPARQL. BioMoby Converter a été développé pour être compatible avec Protégé 3.4 ; nous avons donc décidé d'utiliser l'API Jena. En nous appuyant sur des fichiers OWL ou des fichiers convertis en OWL à partir d'un

fichier OBO, nous avons dû créer deux méthodes de parcours de fichiers différents car un fichier OWL converti de l'OBO comporte quelques spécificités. Par exemple, les commentaires attachés à une classe de l'ontologie sont stockés dans une balise n'existant pas dans le langage OWL. Nous avons créé une méthode permettant de déterminer la nature du fichier OWL afin de déterminer quelle méthode de parcours utiliser.

Les classes de l'ontologie sont parcourues et les informations de chaque classe sont stockées dans un objet Java provenant de jmoby, la librairie Java de BioMoby. A la fin de cette étape, nous possédons un vecteur d'objets jmoby.

6.5.2. Definition des règles de transformation d'une ontologie OWL vers une ontologie BioMoby

Une ontologie OWL est beaucoup plus expressive qu'une ontologie de types de données BioMoby qui est limitée à 3 types de propriétés. Il a fallu tenir compte de cette différence d'expressivité lors de l'implémentation de l'étape de transformation. Lors de cette transformation, chaque **classe OWL** est transformée en type de données BioMoby. Dans le langage OWL on distingue les propriétés selon qu'elles relient des individus à des individus (propriétés d'objets) ou des individus à des types de données (propriétés de types de données). Une **propriété** n'est pas créée physiquement comme un type de données Biomoby.

Les restrictions OWL

Il est possible de contraindre l'image d'une propriété en fonction de contextes particuliers. Cela est réalisé à l'aide de restrictions. Une restriction correspond à l'utilisation d'une propriété pour restreindre la définition d'une classe. Le langage OWL comporte 4 types de restrictions nommées *allValuesFrom*, *SomeValuesFrom*, *hasValue* et *cardinality*.

Dans la Figure 27, les classes *Séquence_ADN* et *Nucléotide* sont reliées par la propriété *composéDe*. Dans le cas d'une restriction OWL de type ***allValuesFrom*** (flèche rouge), cela signifierait qu'une séquence d'ADN est composée **uniquement** de nucléotides. Dans le cas d'une restriction de type ***someValuesFrom*** (flèche verte), cela signifierait qu'une séquence d'ADN est composée d'**au moins un** nucléotide. En raison du manque d'expressivité de l'ontologie de types de données BioMoby, ces 2 restrictions seront utilisées de la même façon et déboucheront donc à la création du même conteneur (flèche bleue).

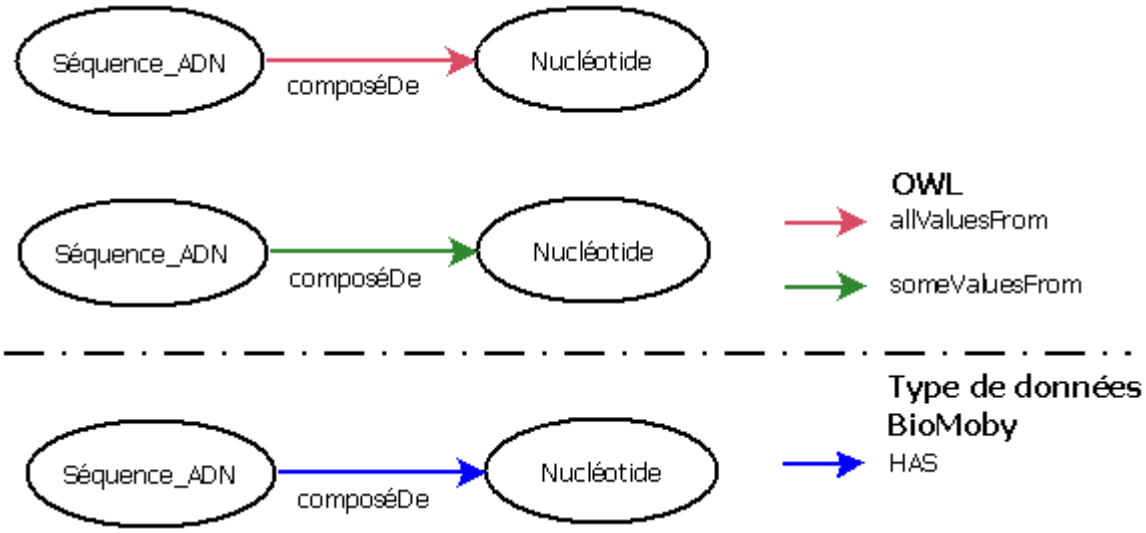


Figure 27 : Prise en compte des restrictions *someValuesFrom* et *allValuesFrom* dans la transformation d'une ontologie OWL vers une ontologie de type de données BioMoby

hasValue permet de faire des restrictions de valeur. Cette valeur peut être soit numérique soit liée à un individu. La prise en compte de ce type de restriction dans notre transformation signifierait donc de permettre la liaison d'une classe OWL à autre chose qu'une classe. La notion d'individu n'est pas présente dans l'ontologie de types de données BioMoby. Dans la version actuelle de notre plugin les restrictions *hasValue* conduisent à la création de classes de types de données correspondant à un individu OWL.

cardinality permet de faire des restrictions de cardinalité. Dans l'ontologie de type de données BioMoby la cardinalité est gérée par l'utilisation de conteneurs HAS ou HASA, permettant de différencier des conteneurs de cardinalité 1 des autres conteneurs. Dans le langage OWL la cardinalité peut prendre n'importe quelle valeur. Durant notre transformation nous ne tiendrons compte que des cardinalités de 1. Elles induiront la création d'un conteneur de type HASA. Toute autre cardinalité induira la création d'un conteneur HAS.

6.5.3. Les inconvénients de la transformation

6.5.3.1.Les relations d'héritage

La transformation des relations d'héritage est simple quand une classe OWL ne possède qu'un seul parent. Dans ce cas, la relation d'héritage OWL est conservée en relation d'héritage dans BioMoby. BioMoby ne supporte pas la gestion de parenté multiple. Une classe ne peut donc pas posséder plus d'une classe parent. Dans le cas ou une classe OWL possède plusieurs parents, une des relations de parenté est créée sous la forme d'une relation

ISA sous BioMoby et les autres sont créés sous la forme d'un conteneur HASA auquel on donnera le nom *hasParent*.

6.5.3.2.La perte de l'espace de nom

Chaque concept d'une ontologie OWL est défini par un espace de nom donnant une indication précise des vocabulaires spécifiques utilisés. Les classes de l'ontologie de types de données BioMoby ne possèdent pas d'espace de nom permettant de décrire un type de données de manière unique. Le nom d'une classe enregistrée correspondra au nom d'une classe OWL amputée de son espace de nom. BioMoby ne supporte pas l'enregistrement de types de données ayant le même nom. L'enregistrement de classes OWL ayant le même nom mais ayant des espaces de nom différents n'est donc pas possible. Afin de dépasser cette limitation, nous avons ajouté la possibilité de faire précédé le nom de chaque classe d'une ontologie par un préfixe défini par l'utilisateur.

6.5.3.3.Les paramètres d'un type de données BioMoby

Une classe de l'ontologie de types de données BioMoby est implémentée sous la forme d'un objet Java possédant plusieurs paramètres. La majorité de ces paramètres sont remplis en prenant compte des champs informés par l'utilisateur de BioMoby Converter. Certains paramètres sont remplis automatiquement. C'est le cas des paramètres correspondant à la relation de parenté, aux conteneurs, à la description de la classe et au LSID.

Le paramètre description d'un type de donnée est rempli automatiquement lors de la transformation en détectant la valeur de la balise *rdfs:comment* d'une classe OWL. Si cette balise n'a aucune valeur associée, un commentaire précisant que la classe a été créée automatiquement est ajouté au type de données correspondant.

Un LSID (Life Science Identifier) (S. Martin et al. 2005) est un moyen de nommer de manière unique et de localiser une information biologique sur le Web. Le LSID des classes que nous créons suit la logique des classes créées manuellement dans BioMoby c'est-à-dire :

urn:lsid:biomoby.org:objectclass:*nom_type_de_données*:*année-jour-mois*T*heure-minute-seconde*Z

6.5.3.4.Le stockage des correspondances

La différence d'expressivité entre les deux langages ontologiques entraine une perte très importante de sémantique lors de la transformation de l'ontologie OWL initiale en ontologie de type de données BioMoby. Nous avons décidé de créer une base de données répertoriant les correspondances entre les classes OWL et les classes BioMoby afin de

pouvoir facilement accéder à la sémantique présente dans l'ontologie initiale lors de la manipulation de types de données BioMoby.

6.5.3.5.Le tri des classes de l'ontologie de types de données BioMoby

Une classe de l'ontologie de type de données Biomoby peut être créée si sa classe parent est déjà présente dans l'ontologie. Dans le cas ou son parent n'est pas déjà présent dans l'annuaire central, l'enregistrement de l'ontologie est interrompue. De la même façon, il n'est pas possible de supprimer une classe d'un annuaire central s'il possède un descendant. De plus, un type de données ne peut pas être créé s'il possède un conteneur (propriété HAS ou HASA) qui réfère à une classe qui n'existe pas dans l'ontologie. Dans ce cas, il faut d'abord enregistrer la classe référant à la propriété. De la même façon, il n'est pas possible de supprimer une classe si celle ci est utilisée comme conteneur dans une autre classe. L'enregistrement et la suppression de classes de l'ontologie de types de données BioMoby nécessitera donc une phase préliminaire de tri, réalisée en créant un objet Java regroupant l'ensemble des classes sous la forme dans graphe acyclique orienté (DAG) dans lequel chaque nœud représente une classe de type de données et chaque arc représente une propriété BioMoby comme c'est le cas dans la Figure 20. Avec une telle représentation, il est possible de trier l'ordre d'enregistrement des classes. Ce tri consiste à parcourir le DAG et à détecter tous les nœuds n'ayant aucune arrête dirigée vers un autre nœud. Ces nœuds seront alors ajoutés à une liste chainée Java de type FIFO (First In First Out), servant de liste d'enregistrement. Comme son nom l'indique, les premières classes ajoutées à cette liste seront les premières à être enregistrées dans BioMoby. Le nœud ajouté à la liste d'enregistrement, ainsi que toutes les arrêtes orientées vers ce nœud est ensuite supprimé. La Figure 28 présente l'évolution du graphe de la Figure 20 lors des différentes étapes de l'algorithme de tri. La Figure 29 présente quand à elle l'ordre d'enregistrement des classes de l'ontologie de type de données dans BioMoby.

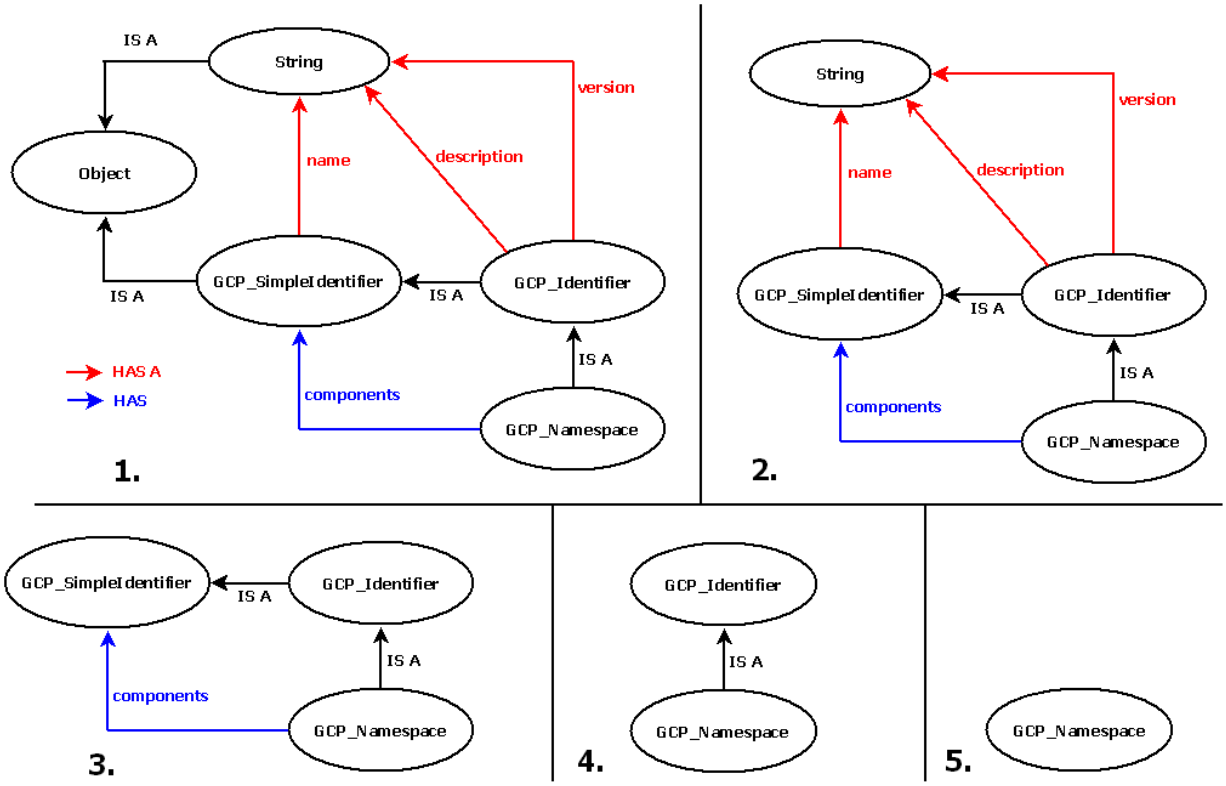


Figure 28 : Evolution d'un graphe lors du tri de ces nœuds pour l'enregistrement d'une ontologie dans BioMoby

Etape 1Object
Etape 2String
Etape 3 GCP_SimpleIdentifier
Etape 4GCP_Identifier
Etape 5GCP_Namespace

Figure 29 : Ordre d'enregistrement des types de données correspondants aux nœuds de la Figure 28

6.6. Les résultats

Pour accéder à BioMoby Converter, il suffit de le charger dans Protégé, d'aller dans la fenêtre correspondante, de sélectionner un annuaire BioMoby et une action à réaliser puis de valider son choix. Le plugin peut être utilisé pour enregistrer une ontologie ou une portion d'ontologie OWL, supprimer une portion de l'ontologie de types de données BioMoby ou encore exporter l'ontologie de types de données BioMoby vers une ontologie OWL.

6.6.1. Le plugin BioMoby Converter

BioMoby Converter se présente sous la forme d'une interface graphique conviviale. Cette interface graphique est présentée dans la Figure 30. Cette interface est découpée en 4 parties. La partie de gauche correspond à l'arborescence de l'ontologie OWL chargée dans Protégé. Il s'agit de la même arborescence que celle présente dans la fenêtre principale de Protégé, limitant ainsi le délai de prise en main de l'outil. La partie de droite représente l'arborescence de l'ontologie de types de données BioMoby. Cette arborescence permet à l'utilisateur de visualiser les types de données déjà présents dans un annuaire sans avoir à lancer *Dashboard*. La partie centrale de l'interface est divisée en 2. La partie supérieure représente un formulaire dans lequel un utilisateur peut sélectionner l'action qu'il souhaite réaliser puis de remplir les champs nécessaires et de valider cette action. La partie inférieure est une console affichant différents messages à l'utilisateur. Une présentation détaillée de l'utilisation possible de chaque élément de l'interface graphique est disponible dans les annexes du mémoire.

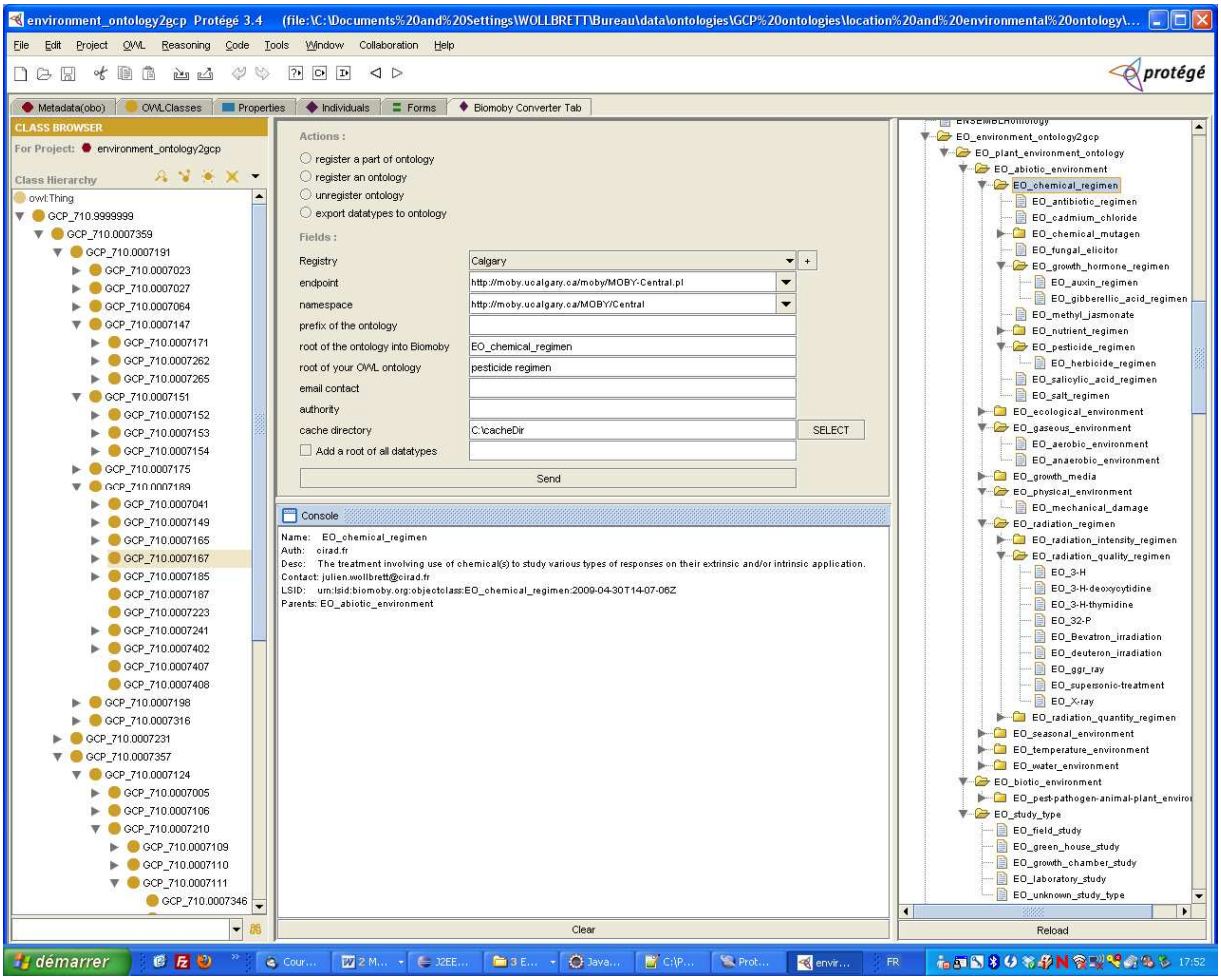


Figure 30 : Interface graphique du plugin BioMoby Converter

6.6.2. Evaluation des performances d'enregistrement

Le Tableau 2 compare le temps nécessaire entre une approche d'enregistrement manuelle sous Dashboard et l'enregistrement automatique réalisé sous BioMoby Converter. Dans ce tableau nous considérons que le temps nécessaire à l'enregistrement manuel d'un type de données est de 45 secondes. L'enregistrement automatique est entre 26 et 54 fois plus rapide qu'un enregistrement manuel. L'enregistrement via BioMoby, en plus d'être plus rapide, permet également d'éviter les erreurs dues à une intervention humaine routinière comme par exemple les fautes de frappes ou la sélection d'un mauvais type de données pour les 3 types de propriétés.

nom de l'ontologie et nombre de classes	temps nécessaire à l'enregistrement manuel (en minutes)	temps nécessaire à l'enregistrement automatique (en minutes)	ratio (manuel/ automatique)
Sequence Ontology (1408 classes)	1056	38,5	27
ontologie de domaine GCP (236 classes)	177	3,25	54
ontologie d'acides aminés (48 classes)	36	1,5	24
ontologie Pizza (96 classes)	72	2	36

Tableau 2 : Temps d'enregistrement sous BioMoby Converter en comparaison d'un enregistrement manuel

6.6.3. Evaluation des autres performances

Le temps nécessaire à la suppression de types de données est identique au temps d'enregistrement. Cela s'explique par le fait que les mêmes étapes préliminaires de tri sont réalisées pour un enregistrement et pour une suppression. Ce temps ne peut pas être comparé avec un temps de suppression manuel car Dashboard ne propose pas cette option.

Le temps nécessaire à l'export de l'ontologie de types de données BioMoby vers une ontologie OWL est très rapide. Cette vitesse peut s'expliquer par l'utilisation d'un dossier de cache qui évite toute communication avec l'annuaire central BioMoby lors de cette étape.

6.7. Discussion

BioMoby Converter permet de répondre à tous les besoins souhaités lors de sa phase de conception. En plus d'automatiser l'enregistrement de types de données, BioMoby Converter permet d'utiliser des bioontologies, considérés comme des vocabulaires standards, pour typer les données consommées et renvoyées par un SW.

6.7.1. Les limites de BioMoby

Le passage d'une ontologie OWL vers une ontologie de types de données BioMoby induit une perte de sémantique importante. Cette perte de sémantique est due à la faible expressivité de l'ontologie BioMoby ne permettant que la création de 3 types de propriétés (ISA, HAS, HASA). Par exemple, des types de restrictions de classe OWL ayant des sens différents sont représentés de façon identique suite à la transformation.

Actuellement chaque utilisateur de données créé ses propres types de données BioMoby et les utilise pour créer ses propres SW. Ceci entraine la création de plusieurs types de données n'ayant pas le même nom mais correspondant au même type de données. La sémantique des SW BioMoby est sensée faciliter leur composition. Cette composition semi-automatique de SW BioMoby est rendue possible par l'utilisation de types de données communs entre la sortie d'un SW et l'entrée d'un autre SW. L'utilisation de types de données différents pour créer des SW réalisant des actions identiques empêche donc cette composition. Il serait possible d'imaginer détecter 2 classes identiques dans l'ontologie BioMoby en recherchant des classes partageant les mêmes propriétés. La sémantique de cette ontologie est basée sur 3 propriétés et la possibilité de typer deux d'entre elle avec du texte libre. Cela n'est malheureusement pas suffisant pour détecter de manière convaincante l'équivalence entre classes.

L'avantage principal de BioMoby est d'offrir toute la machinerie nécessaire à la création automatisée de Services Web. Pour ce faire, une structure de message a été inventée. Cette structure est utilisée dans l'invocation des SW mais également dans les messages de réponse. Cela pose une limitation supplémentaire à la plateforme BioMoby en forçant les clients à être plateforme spécifique.

6.7.2. Remise en cause du choix de BioMoby

Notre choix initial d'utilisation de BioMoby pour créer des SW était guidé par son utilisation pour automatiser la création de certains adaptateurs de la plateforme d'intégration

de données GCP Pantheon. Nous avons donc choisi d'intégrer pleinement BioMoby dans l'architecture de notre plateforme de création automatisée d'adaptateurs. Cependant lors du développement de BioMoby Converter nous nous sommes rendu compte que BioMoby possédait de nombreuses limites. Ses trois grandes limites sont la faible sémantique de son « ontologie », l'obligation pour les clients d'être spécifiques à la plateforme, et le fait de ne pas s'appuyer sur des standards du Web Sémantique (RDF, OWL, SPARQL).

Nous pensions combler le manque de sémantique de son ontologie par le stockage des correspondances entre une ontologie OWL et l'ontologie de types de données BioMoby. Mais nous voulions surtout créer une approche générique pouvant être utilisée par plusieurs plateformes d'intégration. L'obligation des clients à être plateforme spécifique est donc très limitant. BioMoby a été développé à une époque ou le Web Sémantique n'était pas encore mature ce qui lui a permis d'être largement utilisé dans le domaine biologique malgré l'utilisation de messages spécifiques. Actuellement le Web Sémantique permet de créer des solutions génériques en se basant sur des formats considérés comme des standards.

Nous avons donc décidé de nous libérer des limites imposées par l'architecture de BioMoby et de créer une architecture la plus générique possible, totalement basée sur des standards du Web Sémantiques et automatisant au maximum la création d'adaptateurs sémantiques.

6.8. Perspectives

Il serait très intéressant de pouvoir coupler l'utilisation de BioMoby Converter avec le plugin DataMaster (Nyulas & Tu 2007) présent dans Protégé. En effet, ce plugin propose de se connecter à une base de données afin d'en récupérer la structure et ainsi de l'enregistrer sous forme d'ontologie au format OWL. Cet outil pourrait être utilisé pour spécifier des correspondances entre une bioontologie existante (ontologie globale) et l'ontologie de la base de données (ontologie locale). Ces correspondances pourraient faciliter la création d'adaptateurs sémantiques spécifiques à la base de données.

BioMoby Converter a été testé avec différentes ontologies OWL pour vérifier de sa généricité. Il est possible d'imaginer une généricité plus vaste du plugin en le rendant compatible avec d'autres formats d'ontologies ou en prenant en entrée des XML Schéma annotés sémantiquement. BioXSD (Kalas et al. 2010) est un projet dont l'objectif est de créer un XML Schéma se voulant le format d'échange de données standard en bioinformatique. Ce schéma est annoté sémantiquement, en suivant les recommandations du W3C (W3C s. d.), à

l'aide de balises SAWSDL (Farrell & Lausen 2007). L'enregistrement du XML Schéma de BioXSD permettrait de rajouter de la sémantique à BioMoby en faisant le lien entre un réel schéma de types de données et des concepts ontologiques.

6.9. Conclusion

BioMoby Converter fait parti intégrante d'une architecture conçue pour automatiser la création d'adaptateurs sémantiques et s'occupe exclusivement de la phase permettant l'intégration d'une ontologie au format OWL vers un annuaire central Biomoby. BioMoby Converter a été développé sous la forme d'un plugin de l'outil d'édition d'ontologies Protégé. De ce fait il profite de tous ses avantages et de tous les plugins déjà existants permettant ainsi de développer/modifier une ontologie OWL, transformer une ontologie OBO en ontologie OWL, visualiser l'ontologie sous forme de graphe et l'enregistrer dans BioMoby à l'aide d'un seul et même logiciel. En plus de permettre l'enregistrement d'une ontologie dans BioMoby, notre outil permet également d'enregistrer uniquement une portion d'ontologie, de supprimer une portion de l'ontologie de types de données BioMoby ou encore d'exporter l'ontologie BioMoby dans un fichier OWL. BioMoby Converter répond donc à toutes les attentes initiales.

Cependant, en raison des limites de BioMoby (sémantique faible, plateforme spécifique, non basé sur les standards du Web Sémantique) nous avons décidé de ne pas l'utiliser dans notre architecture de création automatisée d'adaptateurs sémantiques.

BioMoby Converter peut tout de même être utilisé pour créer plus facilement une arborescence de types de données en profitant de l'interface graphique conviviale de Protégé pour ensuite l'enregistrer automatiquement dans BioMoby.

7. Développement d'une plateforme d'intégration de données

7.1. Introduction

Les Services Web de Pantheon sont créés de manière semi-automatique à l'aide de la plateforme BioMoby qui ajoute de la sémantique aux Services Web et automatise certaines étapes de leur création et déploiement. Malgré l'utilisation de BioMoby, la création d'adaptateurs pour la plateforme Pantheon reste le goulot d'étranglement à l'ajout de nouvelles sources de données. Le travail nécessaire à l'ajout de nouvelles sources de données a dissuadé certains partenaires du projet GCP de rendre leurs sources de données accessibles sur Pantheon. Il était donc important de proposer une approche la plus automatisée possible.

Comme l'utilisation conjointe de BioMoby et de Pantheon ont montré de nombreuses limites, nous avons décidé de créer une nouvelle plateforme d'intégration (Wollbrett et al. 2011; Wollbrett et al. soumis_2011). Une grande quantité de données biologiques sont stockées dans des bases de données relationnelles et sont accessibles uniquement via des interfaces web. Nous nous sommes donc orientés vers la création d'une plateforme permettant d'accéder à des données issues de bases de données relationnelles réparties. Pour favoriser l'intégration automatique de nouvelles sources de données nous avons décidé de nous appuyer sur les standards du Web Sémantique et les Services Web.

Le Web Sémantique fournit une structure standard permettant de partager des données et de les réutiliser entre différentes applications. Par ailleurs, les Services Web offrent l'avantage d'être facilement réutilisés par des consommateurs de données sans qu'ils n'aient à se soucier de la plateforme d'implémentation utilisée. Il est également possible de réaliser une composition de Services Web existants, de manière à créer des flux de travaux complexes.

L'utilisation conjointe de ces deux technologies offre la promesse d'atteindre l'intégration et l'interopérabilité des ressources bioinformatiques hétérogènes actuellement disponibles sur le Web (M. D. Wilkinson et al. 2009).

7.2. Architecture de la plateforme

L'architecture globale de la plateforme est présentée dans la Figure 31. Cette plateforme permet de créer, de détecter et d'utiliser des adaptateurs de type SWS pour intégrer le contenu de bases de données relationnelles.

Une ontologie locale RDF est créée pour chaque base de données relationnelle à intégrer. Cette ontologie locale que nous appellerons vue RDF, est ensuite annotée à l'aide de concepts ontologiques issus d'ontologies de domaines bioinformatiques. Cette étape d'annotation permet de faire le lien entre le schéma local de chaque base de données relationnelle et le schéma global représenté par les bioontologies. Chaque vue RDF créée puis annotée est stockée dans un annuaire de vues RDF.

Les vues RDF contenues dans l'annuaire sont utilisées pour créer des SWS contenant une requête SPARQL. Plus précisément, ce sont les annotations sémantiques de ces vues qui sont utilisées pour créer des requêtes SPARQL renvoyant des données homologues issues de bases de données relationnelles hétérogènes.

Les SWS sont ensuite enregistrés dans un annuaire de SWS, à partir duquel ils pourront être facilement détectés par des clients. Ces clients pourront ainsi utiliser les SWS comme adaptateurs dépassant ainsi l'hétérogénéité présente dans les schémas des bases de données relationnelles.

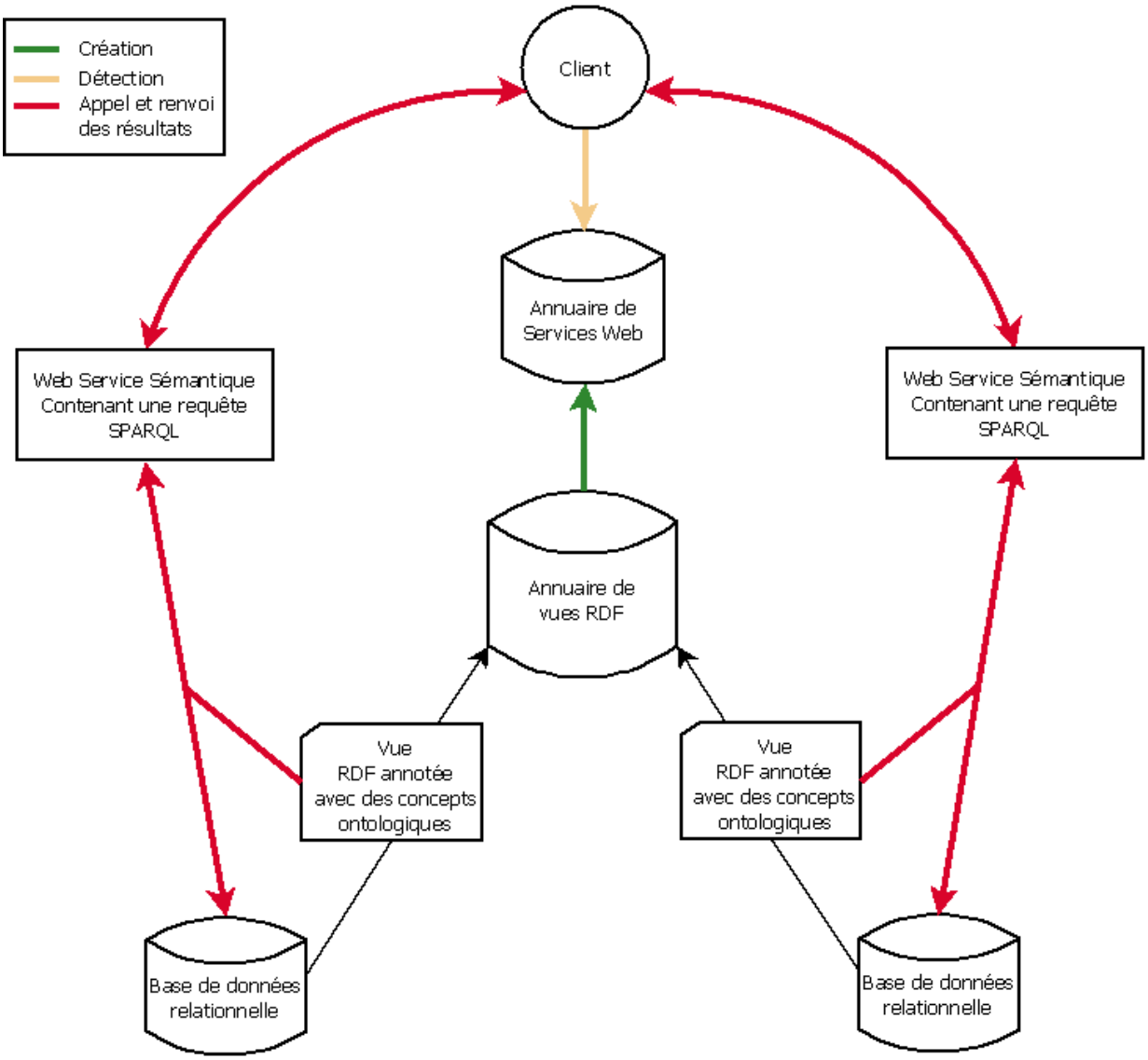


Figure 31 : Architecture globale de la plateforme d'intégration

Notre travail porte sur l'automatisation de l'intégration de nouvelles sources de données dans cette plateforme. Pour cela nous avons développé une application, BioSemantic[12], composée d'une API et d'une interface Web. La Figure 32 présente les parties de l'architecture globale que nous avons tenté d'améliorer. Ces améliorations sont visualisées sous la forme de rectangles noirs. La première contribution de notre travail est la création automatique d'une vue RDF contenant des métadonnées nécessaires à la création automatique de SWS. La deuxième contribution est la création et le déploiement automatique des SWS. Nous sommes actuellement en train de travailler sur l'enregistrement automatique de nos SWS dans l'annuaire de Services Web.

[12] http://southgreen.cirad.fr/?q=content/biosemantic

Ce mémoire ne traitera pas de l'automatisation de l'annotation des vues RDF. Actuellement cette étape est réalisée manuellement.

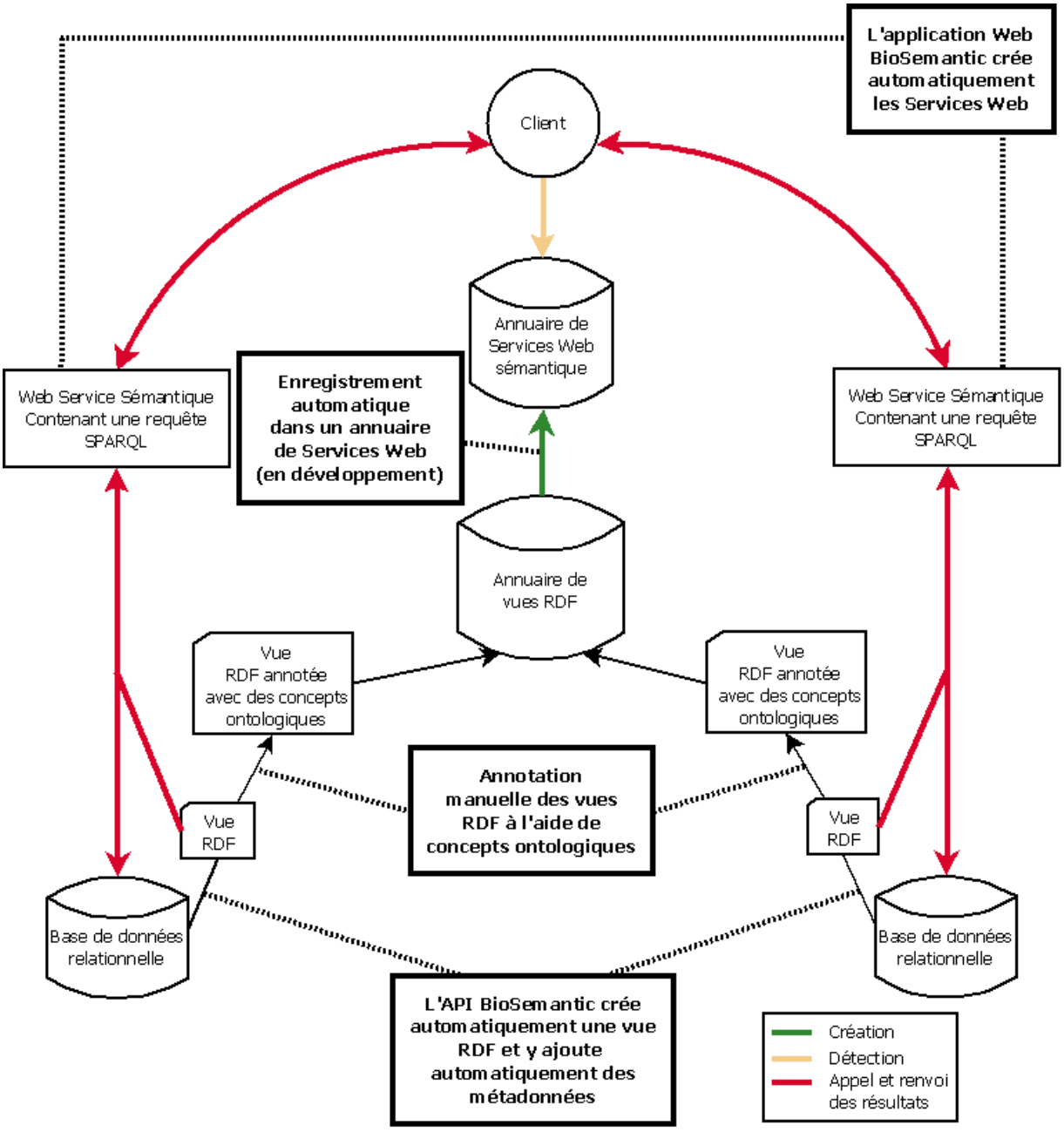


Figure 32 : Contributions de notre travail

7.3. Description des étapes de création de Services Web Sémantiques

La création de SWS est constituée de deux grandes étapes pouvant être réalisées à des instants différents et par des utilisateurs différents. La première étape consiste à détecter des

correspondances entre le schéma local d'une base de données relationnel et le schéma global représenté par des bioontologies. La deuxième étape s'appuiera sur les correspondances entre plusieurs schémas locaux et le schéma global pour créer automatiquement des SWS.

Création de vues RDF annotées

Les différentes étapes de création de correspondances entre les schémas locaux et le schéma global sont schématisées dans la Figure 33.

Dans cette partie, le schéma d'une base de données relationnelle est exporté dans un format RDF à l'aide de la plateforme D2RQ (Christian Bizer 2004); on parle alors de vue RDF du schéma d'une base de données. Cette vue RDF est ensuite enrichie sémantiquement à l'aide de l'API BioSemantic. L'enrichissement sémantique consiste à rajouter automatiquement des métadonnées sur le schéma de la base de données. Il sera présenté en détail dans le chapitre 8. La vue RDF enrichie est ensuite stockée physiquement dans un annuaire de vues RDF. La dernière étape consiste à annoter sémantiquement la vue RDF avec des concepts ontologiques. Ce sont ces concepts ontologiques qui seront utilisés pour créer des SWS. Cette annotation est réalisée manuellement par des experts ayant des connaissances sur le schéma de la base de données à intégrer ainsi qu'une connaissance des bioontologies existantes.

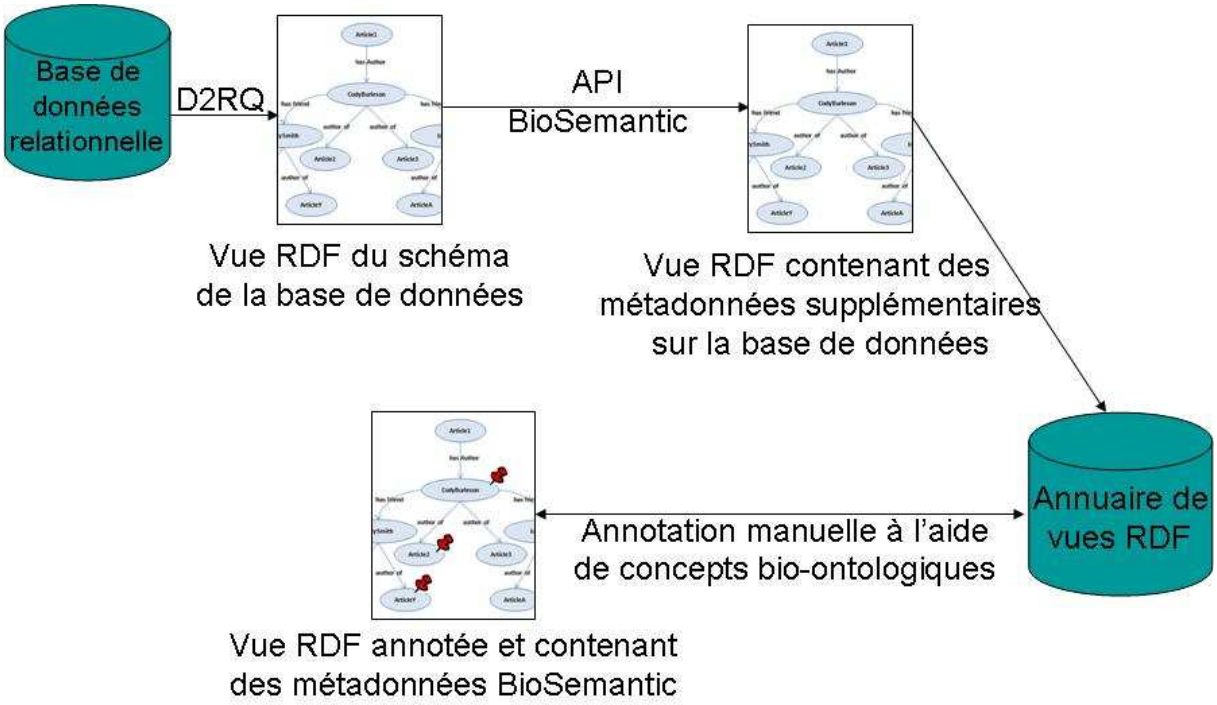


Figure 33 : Etapes de création et d'annotation semi-automatique de la vue RDF de la plateforme BioSemantic

Création automatique de SWS

Les étapes pour la création automatique de SWS sont présentées dans la Figure 34. Dans cette partie, le développeur de SWS n'a pas besoin de connaitre le schéma des bases de données qu'il souhaite intégrer. Il choisit les termes ontologiques qu'il souhaite utiliser comme entrée/sortie de ses SWS. Ces concepts sont utilisés pour parcourir l'annuaire de vues RDF. Si une vue RDF est annotée avec les 2 concepts (entrée/sortie), la plateforme recherche si une requête peut être créée pour cette vue RDF. Une requête peut être créée pour toute vue dont les entrées/sorties sont reliées par un chemin. Si une requête peut être créée, elle est intégrée dans un squelette de Service Web puis un SWS est créé et déployé automatiquement. La sémantique de ces Services Web consiste en une annotation de ces entrées et sorties à l'aide des concepts ontologiques précédemment sélectionnés par le développeur de Services Web facilitant ainsi leur détection et leur composition.

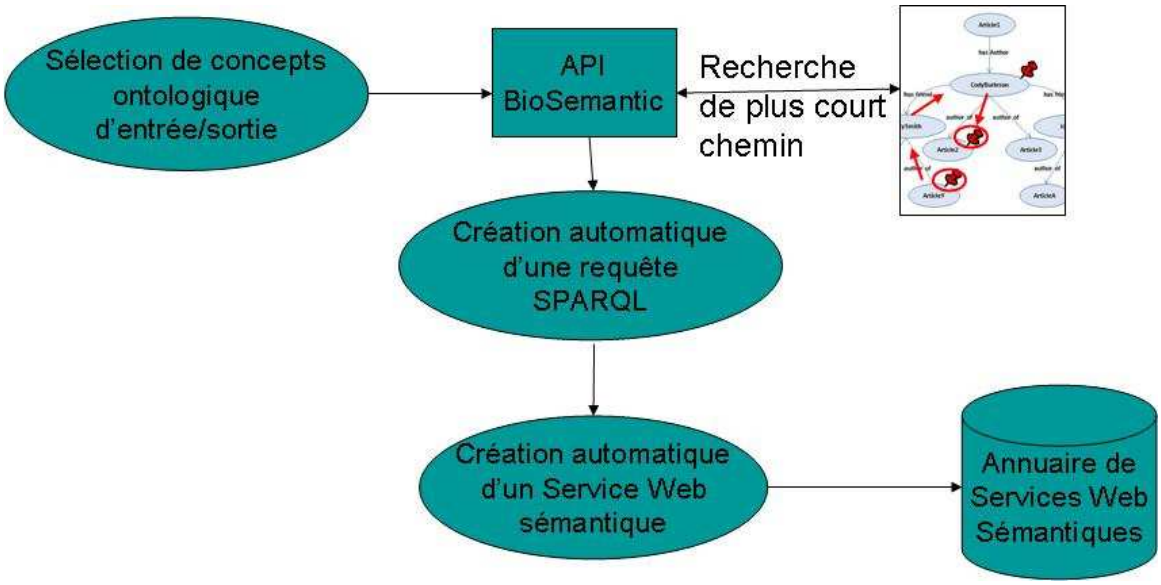


Figure 34 : Etapes de génération automatique de Services Web Sémantiques de la plateforme BioSemantic.

7.4. Création et annotation de la vue RDF

Un grand nombre d'approches ont été mises en œuvre pour faciliter la publication du contenu d'une base de données sur le web en utilisant les technologies du web sémantique. Il peut s'agir de la transformation du schéma d'une base de données relationnelle vers une ontologie ou du peuplement d'une ontologie à l'aide de données comprises dans une base de

données. Cela est réalisé à l'aide de langages descriptifs, permettant de décrire précisément les correspondances entre un concept ontologique et un élément du schéma d'une base de données.

Dans notre approche, nous ne souhaitons pas dupliquer physiquement les données contenues dans une base de données. Nous allons créer une vue du schéma de la base de données relationnelle. Il sera alors possible d'interroger une base de données relationnelle en posant une requête directement sur cette vue. Une requête exprimée sur cette vue sera transformée pour être interprétée par le SGBD de la base de données relationnelle d'origine. Il s'agit d'une étape de mapping entre une base de données relationnelle et une ou plusieurs ontologies. Dans notre approche ce mapping sera utilisé comme une couche intermédiaire entre l'utilisateur et les données stockées. Cette couche fournit une abstraction du contenu de la base de données, permettant aux utilisateurs de formuler des requêtes expressives sur un domaine d'intérêt, tout en cachant les détails de la source. Ces caractéristiques correspondent exactement aux motivations de l'approche d'accès aux données basées sur une ontologie, présentée dans notre état de l'art. Plusieurs outils existants permettent de répondre à ces besoins. Parmi ces outils, seuls deux sont basés sur des standards que ce soit pour la vue (RDF, XML), ou pour le langage de requête (SPARQL), et permettent de mapper une base de données avec plus d'une ontologie. Il s'agit des outils Virtuoso (Erling & Mikhailov 2006) et D2RQ (Christian Bizer 2004).

Les plateformes Virtuoso et D2RQ proposent toutes les fonctionnalités nécessaires à notre approche:

- un algorithme de transformation de schéma de base de données en un fichier RDF: ce fichier est créé automatiquement et correspond à une version RDF du schéma de la base de données.

- un langage de méta-schéma permettant de représenter des correspondances complexes entre un concept ontologique et un élément du schéma de la base de données.

- une approche de médiation permettant d'interroger une BDR sans avoir à exporter les données: Il est possible d'interroger directement la vue RDF à l'aide d'une requête SPARQL. Cette requête est ensuite transformée en requête SQL pour interroger la base de données d'origine.

- un annuaire RDF: l'annuaire RDF permet de stocker et d'interroger plusieurs vues RDF. Cet élément fait partie intégrante de la plateforme Virtuoso. Dans la plateforme D2RQ il n'y a pas d'annuaire RDF intégré. Une API a cependant été développée pour permettre le stockage des vues RDF dans un annuaire RDF Sesame (Broekstra et al. 2001).

Virtuoso est présent sous 2 versions. Une version gratuite et une version payante. Seule la version payante intègre la possibilité d'interroger des bases de données relationnelles sans exporter les données. Notre approche se voulant générique et constituée uniquement d'outils open-source, nous avons donc décidé d'utiliser la plateforme D2RQ pour réaliser le mapping entre ontologie et base de données relationnelle.

Nous allons dans un premier temps présenter en détail tous les éléments de la plateforme D2RQ. Nous détaillerons ensuite la façon dont nous allons utiliser ces différents éléments créer automatiquement des SWS.

7.4.1. L'appliation D2RQ en détail

L'équipe de C. Bizer, qui a développé D2RQ, s'intéresse à la thématique du mapping entre base de données relationnelle et RDF depuis 2002, avec le projet D2R Map (Christian Bizer 2003). D2R Map est un langage déclaratif permettant de décrire les mappings entre le schéma d'une base de données relationnelle et des ontologies OWL ou RDFS. Ce langage permet de définir des correspondances complexes entre les 2 schémas. Dans D2R Map, le fichier contenant les mappings est créé manuellement et peut être utilisé, par un processeur D2R, afin d'exporter automatiquement les données de la base de données vers du RDF.

Le développement de D2RQ est le fruit de l'expérience acquise lors de la création de D2R Map. Il s'agit d'une plateforme permettant de traiter une base de données relationnelle comme un graphe virtuel RDF. Ce graphe est dit virtuel en opposition aux bases de données de graphes (Angles & Gutierrez 2008). En effet, une base de données RDF contient des instances et les relations entre ces instances. Dans un graphe de base de données RDF, un élément du schéma est représenté par un nœud et une relation par un arc orienté. Dans le graphe virtuel RDF créé par D2RQ, les instances ne sont pas obligatoirement présentes. Il est possible de créer ce graphe virtuel en exportant uniquement le modèle de la base de données relationnelle. Dans D2RQ ce graphe virtuel RDF est appelé fichier de mapping. D2RQ est compatible avec les systèmes de gestion de bases de données (SGBD) MySQL, Oracle, ODBC, PostreSQL et tous les SGBD compatibles avec la norme SQL-92 du langage de requête SQL.

D2RQ permet uniquement l'accès aux données, il n'est pas possible d'utiliser cette plateforme pour modifier des tuples ou en insérer de nouveaux.

L'architecture globale de la plateforme D2RQ est présentée dans la Figure 35. Cette plateforme est composée de 3 parties principales :

Le langage de mapping D2RQ : langage utilisé pour créer la vue RDF de la base de données. Il s'agit d'un langage de mapping déclaratif permettant de décrire les relations entre des ontologies ou des vocabulaires RDFS et un schéma de bases de données relationnel. Ce langage est défini formellement par un RDF-Schema dont l'espace de nom est :

<http://wiwiss.fu-berlin.de/suhl/bizer/D2RQ/0.1#>

Le moteur D2RQ : permettant de créer la vue RDF et de transformer une requête SPARQL en requête SQL. Il s'agit d'un plugin pour Jena et Sesame. Il utilise les mappings pour réécrire en SQL les appels Jena et Sesame puis repasser les résultats vers la couche supérieure de la structure

D2R Serveur : Permettant d'interroger les bases de données relationnelles via le web. Il s'agit d'un serveur http qui peut être interrogé via des clients SPARQL grâce à un endpoint SPARQL, en RDF par des clients de « linked data » ou en HTML via des navigateurs web.

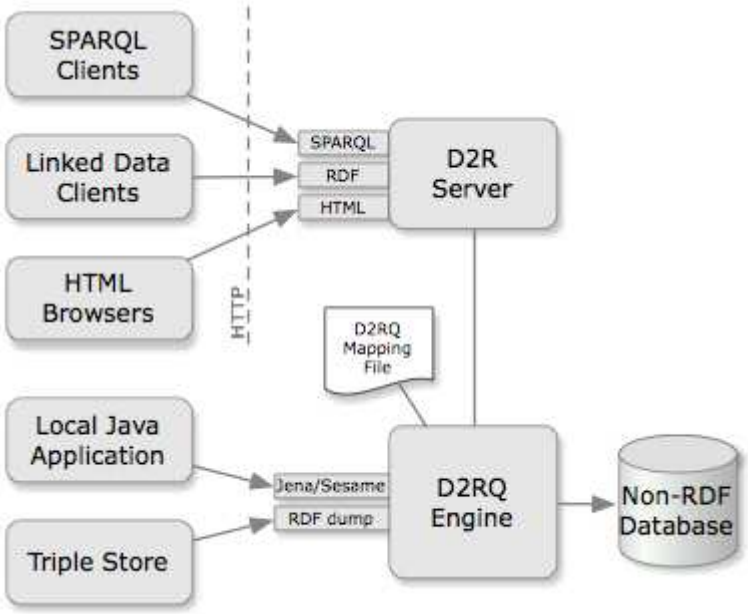


Figure 35 : Architecture de la plateforme D2RQ

Les utilisations de D2RQ en bioinformatique

La plateforme D2RQ a déjà été utilisée avec succès dans le cadre de plusieurs projets en bioinformatique.

L'intégration de données issues de sources distribuées est également un problème majeur pour la bioinformatique dans le domaine biomédical.

L'expression des gènes de *Drosophila melanogaster* est très utilisée en génomique fonctionnelle pour sélectionner des gènes candidats et valider des résultats expérimentaux, permettant ainsi de trouver des traitements pour différentes maladies humaines. OpenFlyData (Miles et al. 2010) propose une interface web de recherche d'expression de gènes chez *Drosophila melanogaster*. Ce projet permet de comparer les données d'expression de gènes contenues dans deux bases de données différentes et rajoute les données d'une troisième source donnant accès aux publications associées à des gènes d'intérêts. Le projet OpenFlyData prouve qu'il est possible de développer une infrastructure d'intégration de données légère qui est utilisable et renouvelable en se basant sur les standards du Web Sémantique. Les sources de données de ce projet sont des bases de données relationnelles possédant chacune leur propre schéma. La plateforme D2RQ a été utilisée pour dépasser l'hétérogénéité de ces schémas et proposer un accès transparent aux données issues des trois sources de données.

Lors de la création de la vue RDF du schéma de la base de données relationnelle, D2RQ propose deux stratégies alternatives. La première approche consiste à réaliser un dump RDF de la base de données contenant le schéma ainsi que les données. La deuxième approche consiste à exporter uniquement le schéma de la source sous la forme d'une vue RDF.

Dans le projet OpenFlyData, l'approche entrepôt a été utilisée pour la totalité des sources intégrées. Le mapping entre les dumps RDF est réalisé à l'aide d'un schéma global. Il s'agit d'une ontologie développée en s'appuyant sur le schéma des sources à intégrer. Cette approche a permis d'intégrer avec succès les trois bases de données relationnelles mais rend difficile l'ajout d'une source de données supplémentaire.

La plateforme D2RQ a également été utilisée dans le projet YeastHub (K.-H. Cheung et al. 2005), développé sous la forme d'un annuaire RDF Sesame contenant des données sur des génomes de levure. Ces données sont issues de plusieurs sources. Un formulaire a été créé, de manière à interroger les données contenues dans l'annuaire. Ce formulaire contient des champs correspondant à des annotations sémantiques ajoutées manuellement dans la vue RDF de D2RQ. L'interrogation d'un champ correspond à l'interrogation d'un dump RDF créé à l'aide de D2RQ. Si l'annotation sémantique est présente dans plusieurs dumps, les données sont renvoyées pour toutes les sources à l'aide de requêtes créées manuellement.

D2RQ a également été utilisé dans le domaine de la neuroscience (Lam et al. 2006). Dans ce projet, une ontologie OWL a été créée manuellement pour chaque base de données. D2RQ est ensuite utilisé pour faire le mapping entre l'ontologie et les données, ce qui permet de peupler les ontologies. Ces ontologies sont alors fusionnées pour obtenir une ontologie unique contenant toutes les données. Le raisonneur Racer (Haarslev & Möller 2001) est ensuite utilisé pour interroger les ontologies à l'aide du langage de requête nRQL (new Racer Query Language) (Wessel & Möller 2005).

7.4.1.1.Quel fichier de mapping choisir : vue RDF ou dump RDF

Les mappings entre un schéma de base de données relationnelle, un vocabulaire RDFS ou une ontologie, est exprimé dans un langage de mapping déclaratif : le langage D2RQ. Ce langage a été conçu pour offrir l'expressivité suffisante à la création de mappings, quel que soit le vocabulaire et la base de données à mapper. En utilisant le langage D2RQ, il est possible de créer 2 types de fichier de mapping différents:

- Une vue RDF du schéma de la base de données. Cela signifie que le schéma de la base de données relationnelle est traduit en RDF. Dans ce cas on parle d'extraction dynamique. Cette vue sera utilisée pour accéder aux instances de la base de données directement dans la base d'origine. La durée d'exécution des requêtes est plus longue en général.
- Un dump de la base de données. On parle d'extraction statique. Dans ce cas le fichier de mapping contiendra les informations sur le schéma de la base de données mais également toutes les instances présentes dans la base lors de la création du dump. Cette approche permet d'accélérer considérablement le temps nécessaire à l'exécution de la requête. En contre partie, l'extraction des données implique que toutes les données ajoutées à la base, après la création du dump, ne seront pas accessibles.

Notre approche ayant pour objectif d'automatiser la création d'adaptateurs pour une plateforme d'intégration de données de type médiateur, nous allons donc utiliser D2RQ pour créer des fichiers de mapping de type vue RDF et ainsi interroger directement les données dans leur source d'origine. Cette vue RDF est sérialisée en N3.

7.4.1.2. La vue RDF de D2RQ

Une vue RDF peut être créée automatiquement par D2RQ. Cette vue RDF contient les informations sur les éléments de la base de données comme les tables, les colonnes, les clés

(primaires et étrangères) ainsi que des métadonnées nécessaires à la connexion à la base de données comme le type de driver utilisé, l'adresse, le nom d'utilisateur et le mot de passe. Cette vue RDF se présente sous la forme d'un fichier RDF composé de triplets. Une ressource est créée pour chaque élément, table ou colonne, de la base de données, puis un prédicat est créé pour chaque relation. L'ensemble de ces triplets forme un graphe orienté. Dans la figure 4, un exemple simple de vue RDF est représenté sous la forme d'un graphe.

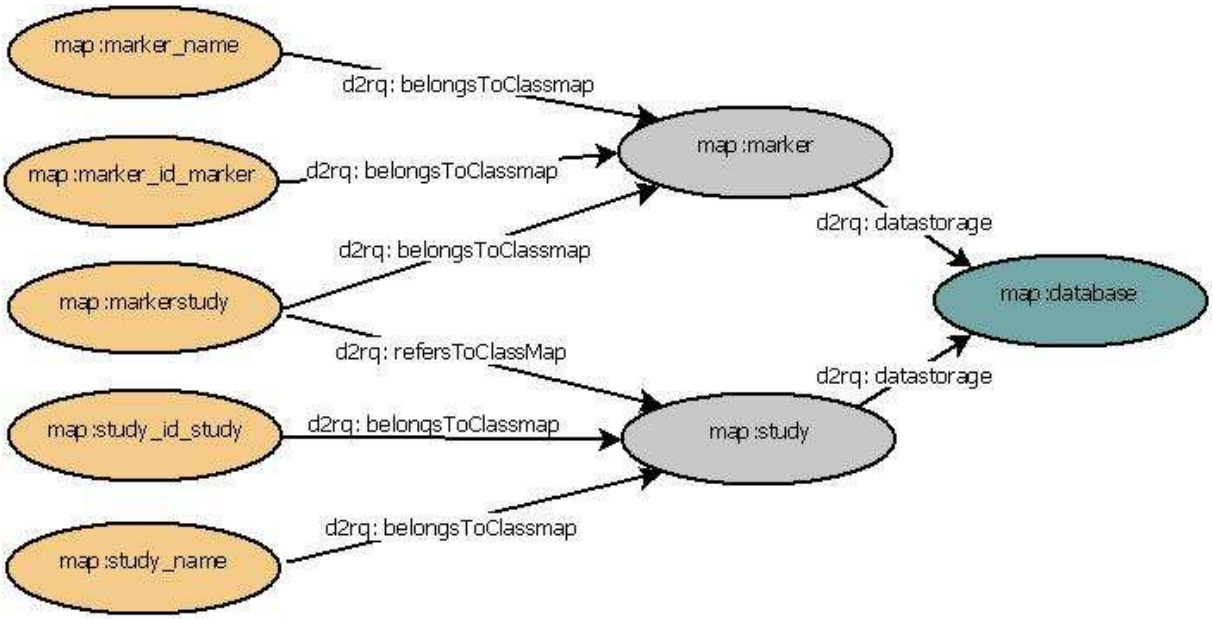


Figure 36 : Graphe représentant une portion de vue RDF

Ce graphe comporte 3 types de nœuds différents.

d2rq:dataStorage: représenté par un nœud bleu. Dans le fichier de mapping un nœud dataStorage représente une base de données. Les informations nécessaires à la connexion y sont associées, comme par exemple le driver à utiliser, l'URL de la base, l'identifiant ou le mot de passe. Ces informations permettront à D2RQ de se connecter à la base de données relationnelle et ainsi de pouvoir lire les instances qu'elle contient.

d2rq:ClassMap: représenté par des des nœuds gris. Une ClassMap correspond à une table. Une ClassMap comporte des informations sur la clé primaire de la table et la base de données à laquelle elle appartient.

d2rq:propertyBridge: représenté par des nœuds orange. Le propertyBridge est principalement utilisé pour définir une colonne. Un propertyBridge comporte des

informations sur la table à laquelle il appartient ainsi que des informations sur les clés étrangères. Une table d'association sans attributs est également représentée à l'aide d'un propertyBridge. Dans ce cas, le propertyBridge comporte plusieurs jointures (d2rq:join) correspondant aux clés étrangères reliant les tables associées.

Le graphe représentant notre vue RDF contient une base de données nommée database. Cette base de données contient deux tables elles même composées de deux colonnes. Le propertyBridge nommé markerstudy correspond à une table d'association sans attributs, permettant de relier les tables marker et study par une association n:n.

Une vue RDF peut contenir des informations sur plusieurs bases de données relationnelles. Fusionner les informations de deux vues RDF revient à fusionner physiquement les deux fichiers RDF correspondant.

Ce fichier de mapping peut être utilisé tel quel ou alors être annoté sémantiquement à l'aide de concepts ontologiques.

7.4.1.3.Le Serveur D2R

Le Serveur D2R est un outil permettant de publier le contenu d'une ou plusieurs bases de données relationnelles sur le Web Sémantique, permettant notamment d'accéder au contenu d'une source via le web des données.

Le serveur D2R utilise le langage de mapping D2RQ pour capturer les correspondances entre le schéma d'une base de données relationnelle et des schémas RDF ou des ontologies OWL. Il permet d'utiliser le dump RDF ou la vue RDF créés par D2RQ pour faire gérer ces correspondances. Les données RDF publiées peuvent être recherchées et affichées sur le Web, ce qui correspond aux 2 paradigmes d'accès aux données spécifiés par le Web Sémantique.

Ce serveur permet d'accéder aux données grâce à 3 interfaces différentes.

L'interface de web des données du serveur D2R rend la description des ressources RDF disponibles en utilisant le protocole http. Une description RDF peut être renvoyée simplement en accédant à l'URI de la ressource sur le web. En effet, tout triplet RDF dont l'objet est une URI peut être vu comme un hyperlien [berners-lee, 2006]. C'est de cette manière que les ressources publiées sur un serveur D2R peuvent être reliées à d'autres bases de données ou à des documents RDF extérieurs. En utilisant des navigateurs du web sémantique comme Tabulator (Berners-lee et al. 2006) ou Disco (Chris Bizer & Gauß 2007),

il est possible de parcourir le web des données en suivant les liens d'une ressource vers la suivante.

L'interface SPARQL permet à des applications de rechercher et requêter les bases de données relationnelles en utilisant le langage de requête SPARQL à travers le protocole SPARQL. Ces requêtes sont exécutées sur un graphe virtuel RDF représentant la base de données relationnelle.

L'interface HTML permet quand à elle d'accéder aux données publiées à l'aide d'un navigateur web classique.

7.4.2. Annotation sémantique de la vue RDF

La création automatique d'une vue RDF ne permet pas de capturer la complexité sémantique du domaine ; elle ne permet pas non plus de dépasser l'hétérogénéité entre schémas. Il est cependant possible d'utiliser cette vue comme une ontologie locale et définir des correspondances entre cette ontologie locale et une ontologie de domaine déjà existante. La mise en correspondances de plusieurs vues à l'aide de mêmes concepts ontologiques permettra ainsi de dépasser l'hétérogénéité locale de chaque schéma de base de données.

La mise en correspondance entre schéma global (ontologies) et schémas locaux (schéma RDF des sources de données) est réalisée en utilisant le langage D2RQ. Il permet d'utiliser des ontologies pour réaliser des mises en correspondance simples, c'est-à-dire lorsque un élément du schéma de la base de données correspond exactement à un concept ontologique. Un exemple de mapping simple est présenté dans la Figure 37. Dans cette figure, on peut voir une annotation sémantique réalisée à l'aide de la balise *d2rq :property*. Cette balise permet d'associer la colonne *name* de la table *marker* avec le concept ontologique *GenomicFeatureDetector*. Il est également possible de réaliser des mappings complexes, grâce notamment aux mappings conditionnels gérés par D2RQ. Un exemple de mapping conditionnel est présenté dans la Figure 38. On y voit une annotation sémantique entre la colonne *name* de la table *study* et le concept ontologique *GenotypingStudy*. La balise *d2rq:condition* ajoute une condition à ce mapping. Dans notre exemple l'annotation réalisée à l'aide de la balise *d2rq:property* n'est valable que pour les tuples dont la colonne *studytype* de la table *study* prend la valeur 'genotype'.

Cela signifie qu'il sera possible d'interroger la source de données en interrogeant le concept ontologique plutôt que l'élément du schéma correspondant.

```
map:marker_name a d2rq:PropertyBridge;  # défini le concept marker_name
d2rq:belongsToClassMap map:marker; # précise qu'il est rattaché à la table marker
d2rq:column "marker.name";          # précise qu'il s'agit de la colonne name de la table marker
d2rq:property gcpdm:GenomicFeatureDetector; # y associe une annotation sémantique
```

Figure 37 : Mapping simple entre l'attribut *name* de la table *marker* et le concept ontologique GenomicFeatureDetector issue de l'ontologie GCP domain model.

```
map:study_name a d2rq:PropertyBridge; # défini le concept study_name
        d2rq:belongsToClassMap map:study; # précise qu'il est rattaché à la table marker
        d2rq:column "study.name";          # précise qu'il s'agit de la colonne name de la table study
        d2rq:property gcpdm:GenotypingStudy;# y associe une annotation sémantique
        d2rq:condition « study.studytype= 'genotype' »;# ajoute une condition à l'annotation
```

Figure 38 : Mapping conditionnel entre l'attribut name de la table study et le concept GenotypingStudy issue de l'ontologie GCP domain model.

Une annotation sémantique dans une vue RDF D2RQ peut être vue comme l'ajout de drapeaux sémantiques à une localisation précise du schéma de la source de données. Il est ainsi possible d'interroger un élément du schéma annoté en utilisant non pas la structure du schéma mais l'annotation sémantique. Il est également possible d'utiliser ces annotations sémantiques pour détecter des correspondances entre des éléments issues de sources de données réparties. La Figure 39 est un graphe représentant deux parties de deux bases de données relationnelles différentes. Chacune de ces parties représente une table (nœuds orange) composée de deux colonnes (nœuds gris). Le rectangle vert représente la clé primaire de la table. Une table s'appelle *marker* et l'autre table *snp*. Un SNP est une variation d'une seule paire de base du génome entre des individus d'une même espèce, et peut être utilisé comme marqueurs. Les colonnes correspondant au nom d'un marqueur et d'un SNP ont toutes les deux été annotées à l'aide du concept ontologique *genomicFeatureDetector*. Il est donc possible d'utiliser ce concept comme un drapeau sur les deux vues RDF. L'interrogation conjointe des deux vues RDF à l'aide de ce concept ontologique, permettra de renvoyer les données comprises dans les 2 sources de données hétérogènes.

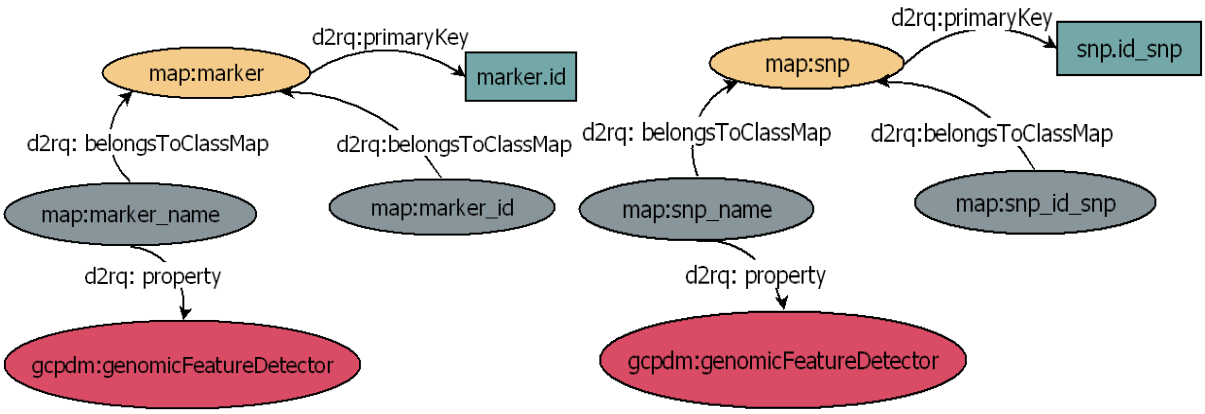


Figure 39 : Utilisation des annotations sémantiques pour détecter des correspondances entre des sources hétérogènes.

Les annotations sont actuellement rajoutées manuellement avec un éditeur de texte. Elles sont réalisées par un curateur expert connaissant le schéma de la source de données et le contexte dans lequel les données ont été créées. Cet utilisateur doit également être familier avec les bioontologies. L'annotation sémantique est la seule étape manuelle de la plateforme d'intégration BioSemantic. Il s'agit également d'une étape très importante car elle nécessite une expertise humaine de qualité. En effet, toutes les étapes de la plateforme BioSemantic vont utiliser ces annotations par la suite. Une mauvaise annotation pourrait donc conduire au renvoi de données non souhaitées (faux positifs) ou au non renvoi de données intéressantes (faux négatifs).

7.4.3. Enrichissement sémantique de la vue RDF

D2RQ utilise son propre langage descriptif pour permettre de publier sur le web des données issues de BDR. La vue RDF créée et annotée permet d'obtenir ce résultat. Cependant, ce n'est pas la seule utilisation de la vue RDF dans notre plateforme. En effet, nous souhaitons également utiliser cette vue RDF d'une manière nouvelle. Ce fichier RDF peut être visualisé sous forme d'un graphe préalablement annoté avec des concepts ontologiques. Ces concepts peuvent donc être utilisés comme des drapeaux pour accéder à un élément de la source sans avoir à connaitre le modèle. Nous souhaitons utiliser ces particularités de graphe et d'annotation sémantique pour parcourir automatiquement ce graphe et rechercher le plus court chemin permettant de relier deux concepts ontologiques précédemment sélectionnés. Ce plus court chemin pourrait ainsi être utilisé pour créer automatiquement une requête SPARQL sans aucune connaissance du schéma de la base.

Ce plus court chemin consiste à passer d'une relation à une autre en utilisant les clés étrangères. Ce passage d'une relation à une autre dans le plus court chemin sera par la suite utilisé pour créer une jointure dans la requête correspondante. La recherche de plus court chemin utilisera les triplets présents dans la vue RDF permettant de décrire les clés primaires, les clés étrangères, l'appartenance d'une colonne à une table ou encore l'appartenance d'une table à une base de données.

Cependant, la vue RDF créée automatiquement par D2RQ manque d'expressivité pour pouvoir créer automatiquement des requêtes. En effet, aucune information sur le type de relations entre les tables n'est prise en compte. Or, le type de relation entre deux entités peut permettre de déterminer la manière dont le graphe doit être parcouru, et donc, la manière dont les requêtes SPARQL doivent être créées. C'est pour cela que nous devons dans un premier temps réaliser une étape d'enrichissement sémantique de la vue RDF. Cet enrichissement est réalisé automatiquement. La logique et les algorithmes utilisés pour enrichir la vue RDF seront présentés en détail dans le chapitre 8 de ce mémoire. Cet enrichissement sémantique conduit à l'ajout de triplets composés de 3 types de prédicats différents.

rdfs:subClassOf : prédicat utilisé pour ajouter des informations sur les relations d'héritage, d'agrégation et de composition. Lors de la recherche de plus court chemin, le parcours d'un tel prédicat conduira à la combinaison de plusieurs plus courts chemins pour créer une requête SPARQL.

dr:associatedTo : prédicat permettant de détecter facilement les tables d'associations, qu'il s'agisse de table d'association avec ou sans attributs. Lors de la recherche du plus court chemin, le passage par une table d'association sera beaucoup moins coûteux que le passage par une autre relation. Le passage par ce type de tables sera donc favorisé.

dr:arity : prédicat utilisé pour ajouter des informations sur l'arité d'une table d'association. Plus l'arité d'une telle table est petite, plus le coût de passage par cette table sera petit. Ce prédicat permettra de favoriser le passage par les tables d'association binaires.

Suite à l'annotation sémantique à l'aide de concepts ontologiques et l'enrichissement sémantique définissant explicitement certains types de relations dans le schéma de la base de données relationnelle, nous disposons d'une vue RDF prête à être utilisée dans la plateforme d'intégration BioSemantic.

7.4.4. Stockage des vues RDF

Nous avons donc décidé de créer un dossier unique contenant toutes les vues RDF. Ce dossier est intégré dans une application web, permettant d'y accéder via internet. La centralisation de ces fichiers RDF va permettre de faciliter leur parcours, et donc faciliter la recherche de correspondances entre les vues RDF hétérogènes.

7.5. Création automatique de Services Web sémantiques

L'intégration des sources de données hétérogènes sera réalisée grâce à la création automatique d'adaptateurs de type SWS. Ces Services seront au format SOAP/WSDL (Erik Christensen et al. 2001; Gudgin et al. 2007) et les annotations sémantiques seront ajoutées à l'aide de balises SAWSDL (Farrell & Lausen 2007) .

La création des SWS se fait de façon asynchrone par rapport à la création et à l'annotation de vue RDF. Il est donc possible d'utiliser toutes les vues RDF annotées précédemment pour créer des SWS. Afin de bien détailler cette étape, nous avons décidé de la découper en 3 sous étapes que nous présenterons séparément. Ces 3 sous étapes peuvent être visualisées chacune d'une couleur différente dans le diagramme d'activité de la Figure 40. La première partie (verte), consiste à détecter les vues RDF susceptibles de permettre la création d'un SWS, la seconde (orange) consiste à créer les différents fichiers nécessaires à sa création et la dernière (grise) consiste à le déployer. Le détail de chaque étape est présenté dans l'annexe D. Dans cette partie nous détaillerons uniquement l'ajout d'annotations sémantiques au fichier WSDL et l'étape d'enregistrement dans l'annuaire de SWS Bio

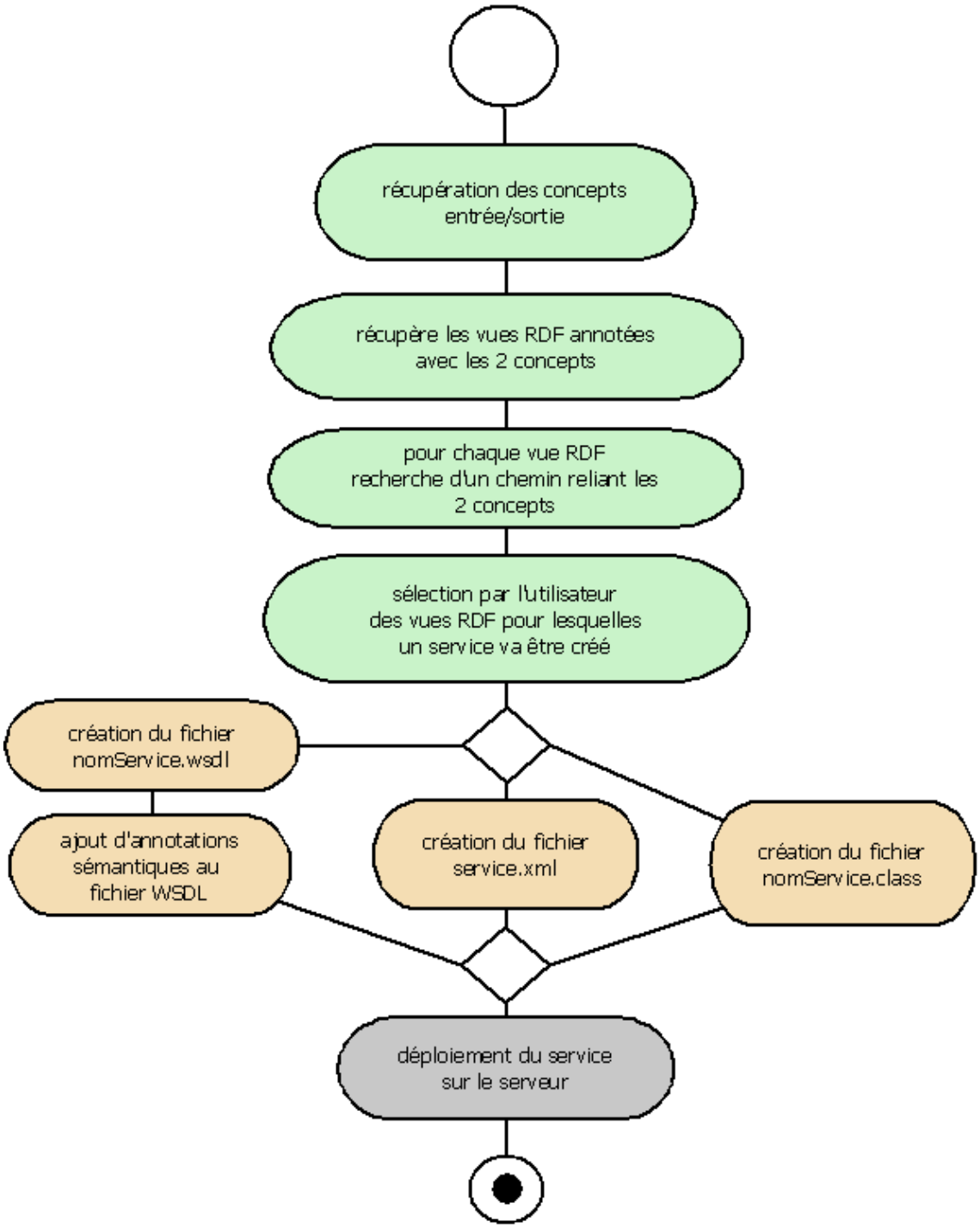


Figure 40 : Diagramme d'activité de l'application web BioSemantic

7.5.1. Annotation sémantique du fichier de description

Les 3 fichiers que nous venons de créer sont suffisants pour créer automatiquement un Service Web. Pour l'annotation sémantique des Services Web, nous avons décidé d'utiliser l'approche recommandée par le W3C, à savoir SAWSDL (Farrell & Lausen 2007).

Les annotations SAWSDL sont intégrées directement dans le fichier WSDL de description du SWS sous la forme de balises supplémentaires. Ces balises, détaillées dans la Figure 41, sont ajoutées automatiquement au fichier WSDL créé précédemment. Dans cette figure, l'annotation SAWSDL est présentée en gras. Dans cet exemple, elle permet d'annoter sémantiquement l'attribut d'entrée de la méthode contenue dans notre SWS à l'aide du concept ontologique correspondant.

```
<xs:element name="method">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="input" nillable="true"  type="xs:string"
      sawsdl:modelReference="http://gcpdomainmodel.org/GCPDM#GCP_GenotypeStudy"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 41 : Exemple d'annotation sémantique de l'entrée d'un Service Web à l'aide d'une balise SAWSDL

7.5.2. Enregistrement dans un annuaire de Services Web

Nous avons décidé d'utiliser un des annuaires principaux de Services Web bioinformatiques : BioCatalogue (Bhagat et al. 2010). Il s'agit d'un annuaire hébergé à l'EMBL-EBI permettant d'enregistrer tout type de Servicse Web spécifiques au domaine de la biologie. Il n'héberge pas les Service Web eux-même mais se charge d'offrir un moyen de les découvrir et de les annoter sémantiquement. BioCatalogue nous permettra d'enregistrer facilement nos SWS et d'utiliser leurs annotations sémantiques pour faciliter leurs découvertes par d'autres utilisateurs.

7.6. Résultats

7.6.1. Utilisation de l'interface de l'application Web BioSemantic

Destinée à faciliter les étapes de création des SWS, l'application Web[13] de la plateforme BioSemantic possède 3 types d'actions. Le détail des interactions entre les différents acteurs pour ces 3 actions est présenté dans la Figure 42. Dans cette figure, la partie verte représente l'étape de création automatique d'une vue RDF contenant des métadonnées nécessaires à la plateforme. Cette vue RDF peut ensuite être téléchargée pour être annotée

[13] Cette application web est développée en Java sous Eclipse avec un déploiement sur le serveur web Apache Tomcat.

sémantiquement. La partie blanche correspond à la phase de depôt de la vue RDF vers l'annuaire RDF de l'application Web. La partie orange de la figure correspond à la phase de création automatique de SWS. Cette figure permet de bien visualiser le niveau d'automatisation lors de ces 3 phases. En effet, seul la phase de création de SWS nécessite une intervention humaine significative.

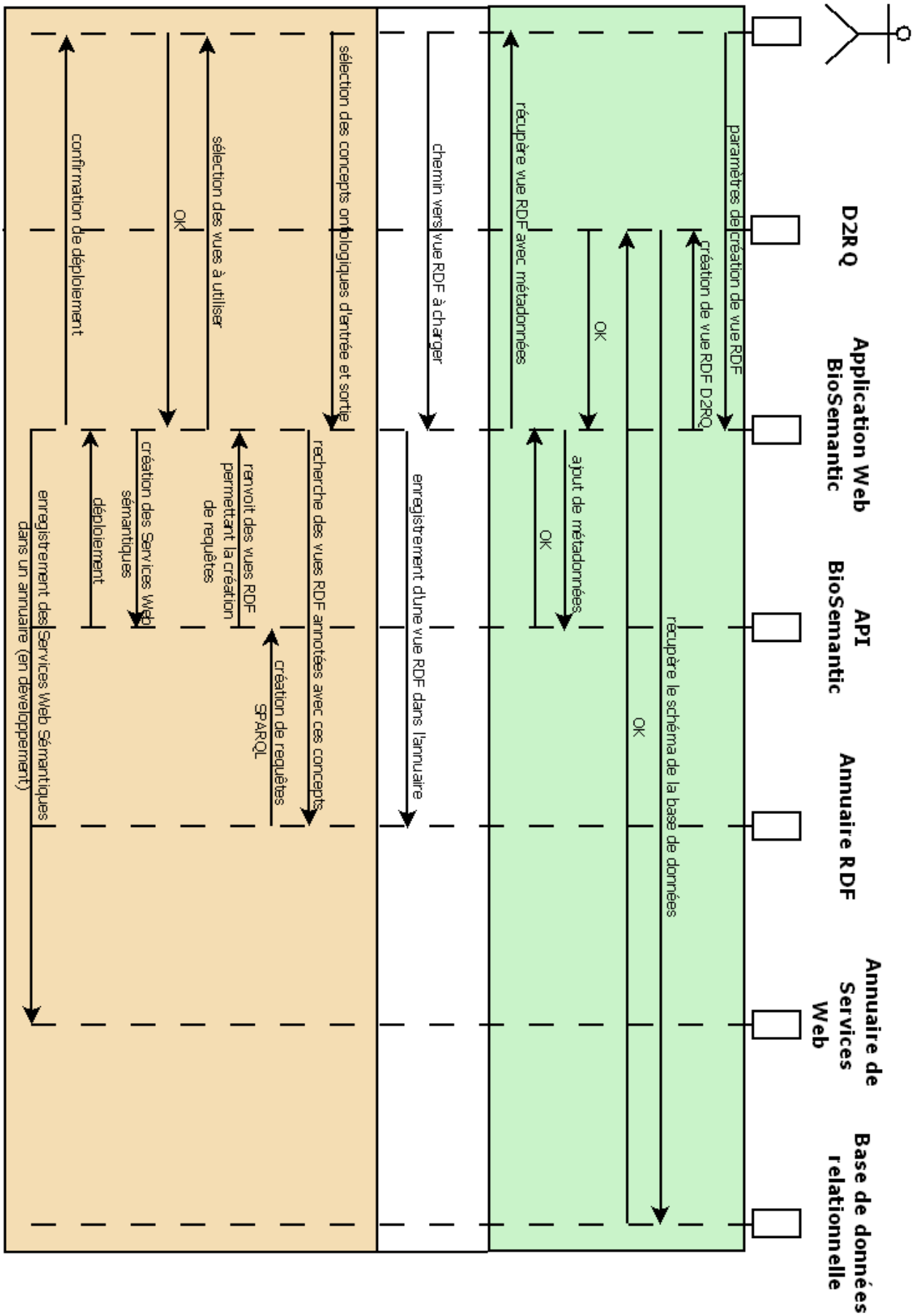


Figure 42 : Diagramme de séquence de l'application Web BioSemantic

7.6.1.1.Création d'une vue RDF

L'étape d'enrichissement sémantique est réalisée automatiquement puis a été intégrée à l'étape de création automatique de la vue RDF par D2RQ. La création d'une vue RDF via D2RQ, est réalisée initialement à l'aide d'une ligne de commande nécessitant plusieurs paramètres. Pour faciliter cette étape, nous avons créé un formulaire HTML contenant un champ pour chaque paramètre. Les différentes interactions entre les éléments du système sont présentées dans la partie verte de la Figure 42. Cette interface, présentée dans la Figure 43, est très simple d'utilisation, la création d'une vue RDF est possible en renseignant uniquement les champs demandés. La vue RDF est créée automatiquement lors de l'envoi du formulaire. Une fenêtre s'ouvre alors proposant de télécharger la vue dans le but de l'annoter à l'aide de concepts bioontologiques.

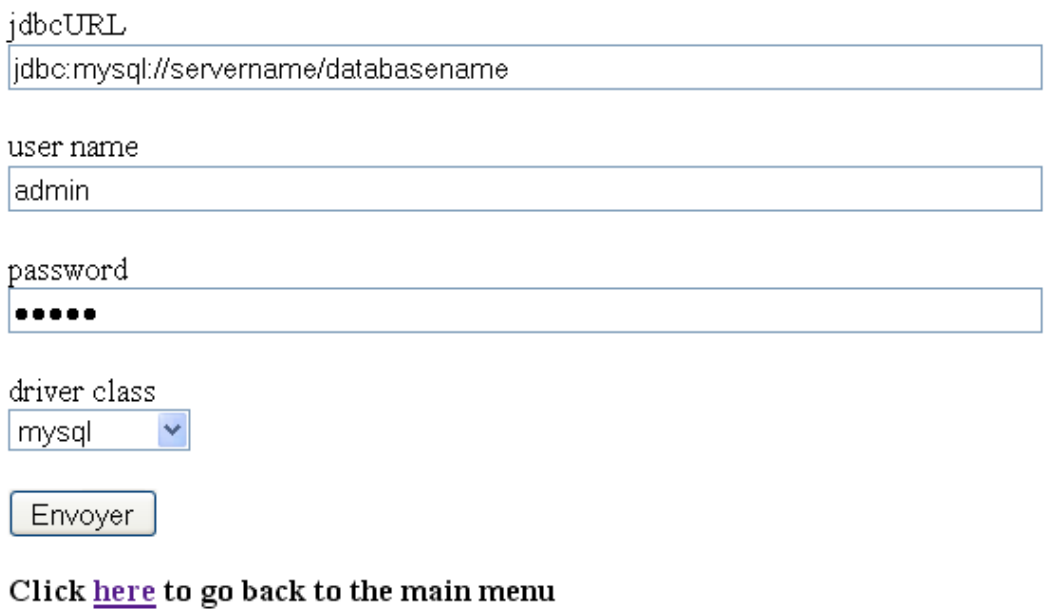


Figure 43 : Interface graphique de création d'une vue RDF enrichie sémantiquement

7.6.1.2.Chargement de la vue RDF dans l'annuaire

L'étape d'annotation sémantique de la vue RDF restant manuelle il est nécessaire de télécharger une vue créée automatiquement. Il est donc nécessaire, après avoir réalisé l'annotation, de charger la vue RDF dans l'annuaire de l'application Web. La Figure 44 montre l'interface graphique permettant de réaliser cette action. Il s'agit d'un formulaire qui permet de sélectionner, le chemin vers la vue RDF. Le fichier est automatiquement chargé lors de la validation.

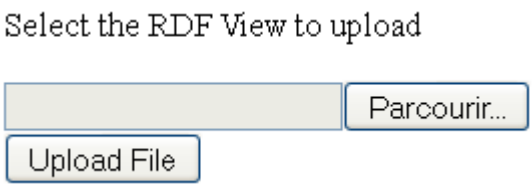


Figure 44 : Interface graphique de chargement d'une vue RDF annotée dans l'annuaire de l'application Web

7.6.1.3.Création de Services Web Sémantiques

L'étape de création des SWS est entièrement automatique. Elle est réalisable via des interfaces graphiques s'affranchissant ainsi de connaissances sur les schémas des bases de données, les formats standards du Web Sémantique (RDF, OWL, SPARQL), ou le langage descriptif D2RQ.

La première étape consiste à sélectionner deux concepts ontologiques. Ces concepts peuvent être sélectionnés par un utilisateur sans aucune connaissance des schémas des sources qu'il souhaite intégrer. L'interface graphique permettant de sélectionner ces concepts est présentée dans la Figure 45. Il s'agit d'un formulaire HTML contenant deux zones de texte : une pour le concept d'entrée et une pour le concept de sortie. Comme nous pouvons le voir sur la Figure 45, les URI des concepts sont utilisées pour les définir.

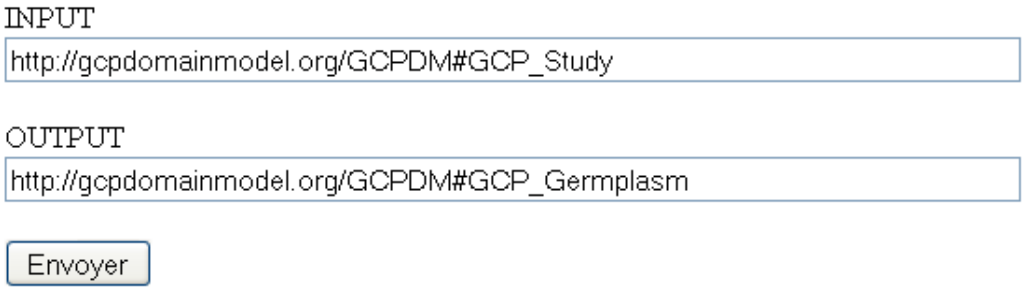


Figure 45 : Interface graphique de sélection des concepts ontologiques d'entrée et de sortie des Services Web Sémantiques à créer

Toutes les vues RDF ayant été annotées avec les concepts ontologiques sélectionnés vont être utilisées pour tenter de créer des requêtes SPARQL. Un SWS peut être créé pour chaque vue RDF permettant de créer une requête SPARQL. L'application Web BioSemantic permet de visualiser ces vues et permet à l'utilisateur de sélectionner celles qu'il souhaite utiliser pour créer un SWS. La Figure 46 montre le formulaire permettant de sélectionner les

vues RDF. Dans cette figure nous avons sélectionné les deux premières vues RDF afin de créer deux SWS.

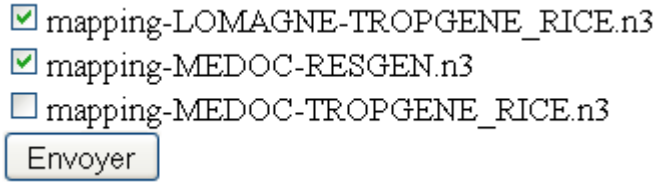


Figure 46 : Interface graphique de sélection des vues RDF utilisées pour créer des Services Web Sémantiques

Une dernière interface graphique permet de confirmer le déploiement des SWS. Cette interface est présentée dans la Figure 47. Cette interface possède également un lien vers le fichier WSDL de chaque SWS créé. Il est ainsi possible de l'afficher et ainsi de vérifier sa structure et la présence des balises d'annotations SAWSDL. Ce fichier WSDL est présenté dans la Figure 48 dans laquelle on peut notamment repérer les annotations sémantiques encadrées en rouge.

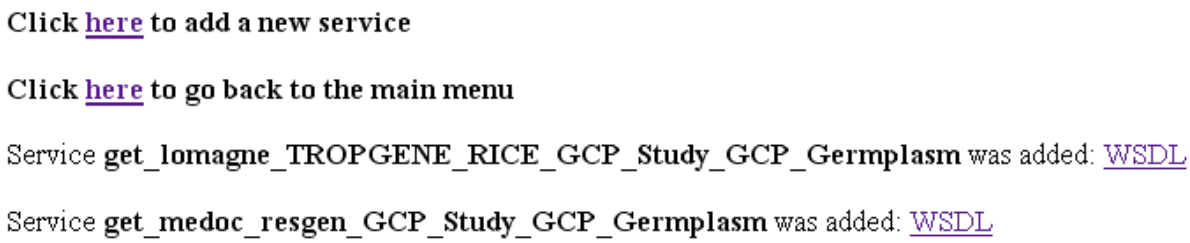


Figure 47 : Confirmation du déploiement des Services Web Sémantiques

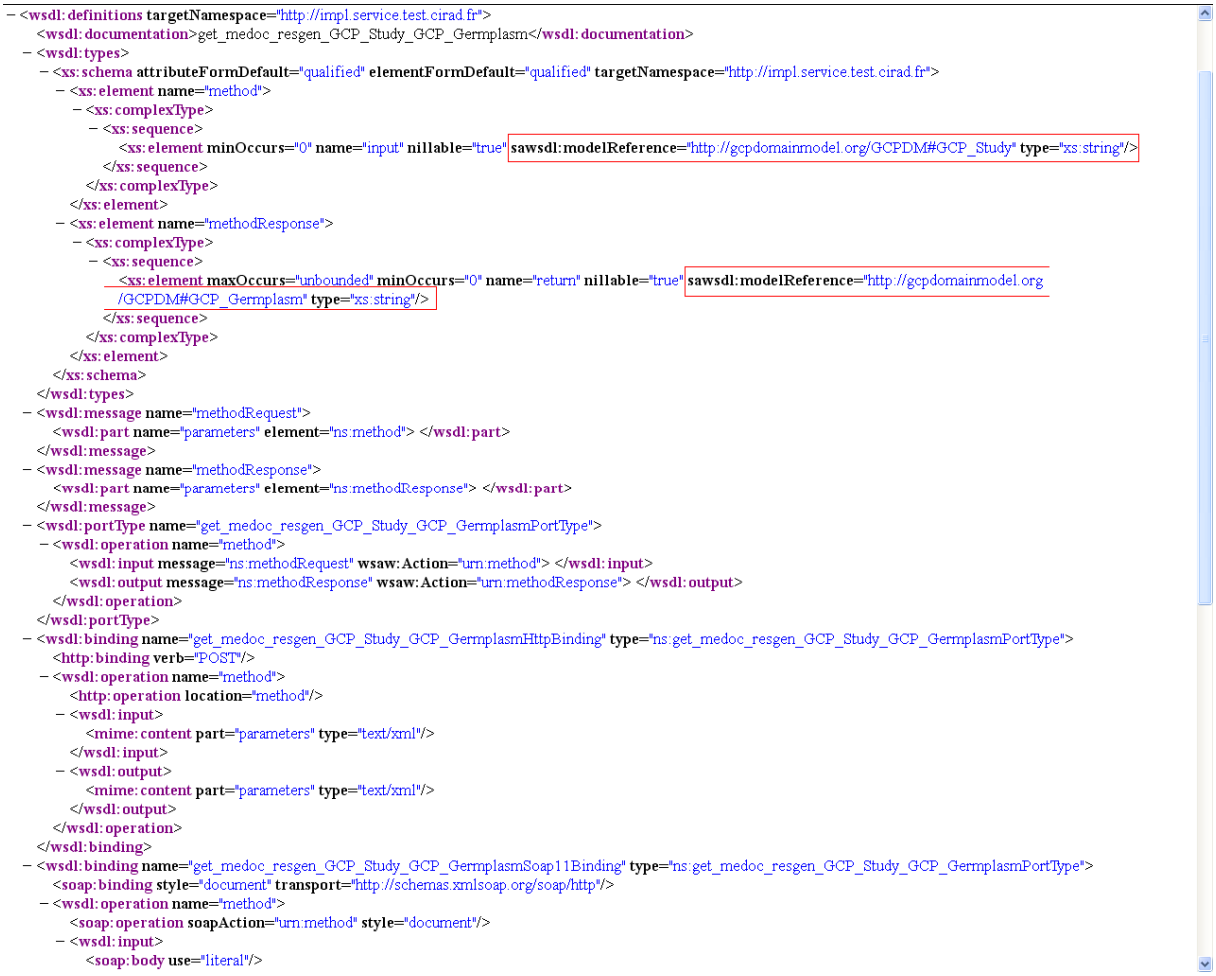


Figure 48 : Affichage du fichier WSDL

7.6.2. Cas d'utilisation pour le fonctionnement de BioSemantic

GenDiversity[14] est une application Web d'analyse de la diversité génétique pour des populations de plantes. Pour cela, elle combine les données de génotypage issues de multiples sources distribuées. Cette application a été développée au CIRAD en support aux analyses de diversité génétique réalisées par des logiciels dédiés. Un des flux de travaux disponibles dans GenDiversity consiste à récupérer les germplasmes et les marqueurs associés à une étude de génotypage. Ces germplasmes et marqueurs sont ensuite utilisés pour renvoyer la diversité allélique.

Recherche de diversité allelique

Pour une analyse standard, un utilisateur de GenDiversity doit d'abord sélectionner une source de données. Cette source de données peut être un fichier Excel issu du *GCP*

[14] http://gendiversity.cirad.fr/

Central Repository ou une base de données relationnelle dans le cas d'un module de TropGene.

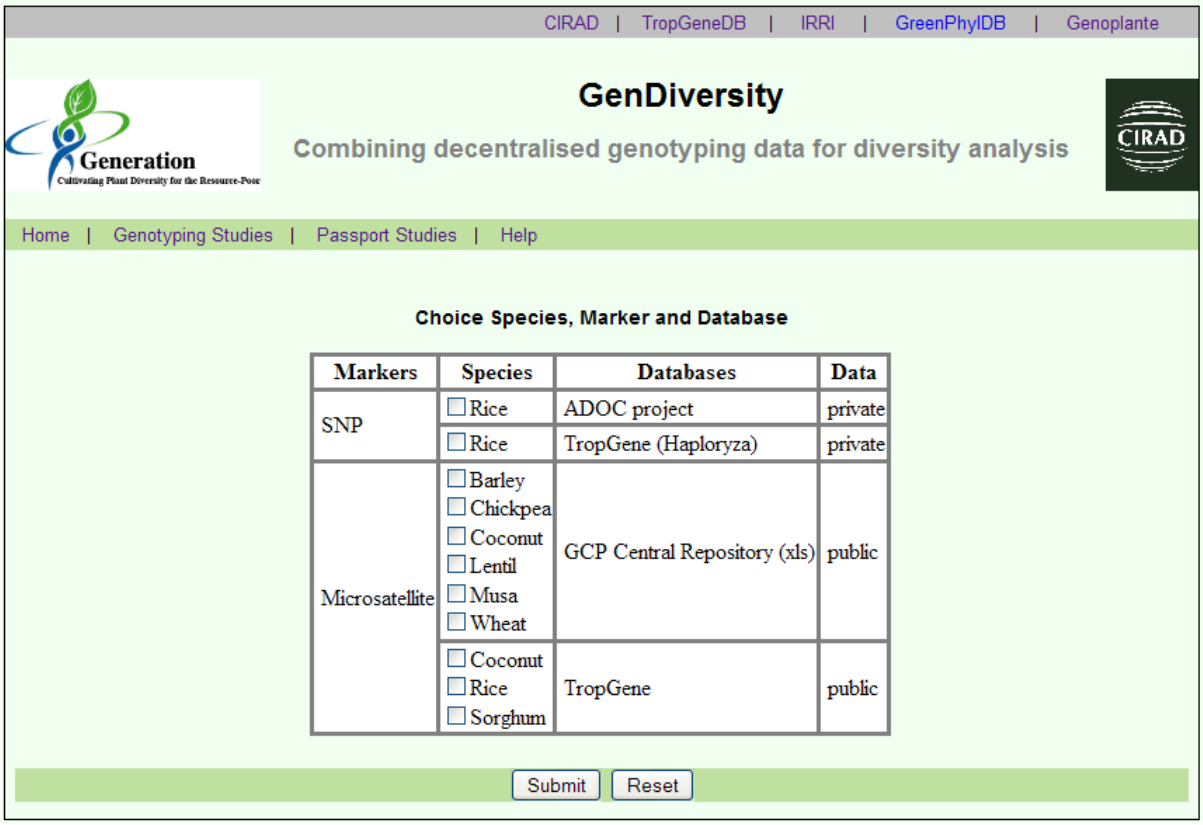


Figure 49 : Fenêtre de sélection de la source de données

L'application récupère ensuite toutes les études de génotypage présentes dans la source de données sélectionnée (Figure 50).

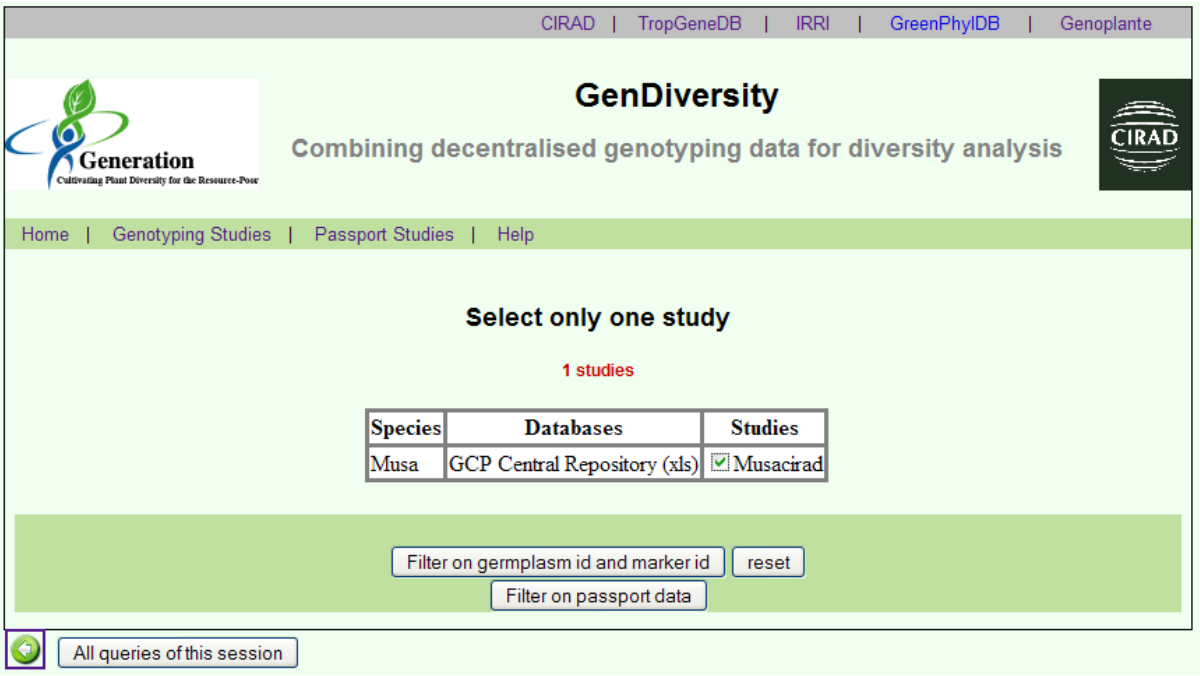


Figure 50 : Fenêtre de sélection d'une étude de génotypage

L'application détecte ensuite les marqueurs et les germplasmes utilisés dans cette étude (Figure 51).

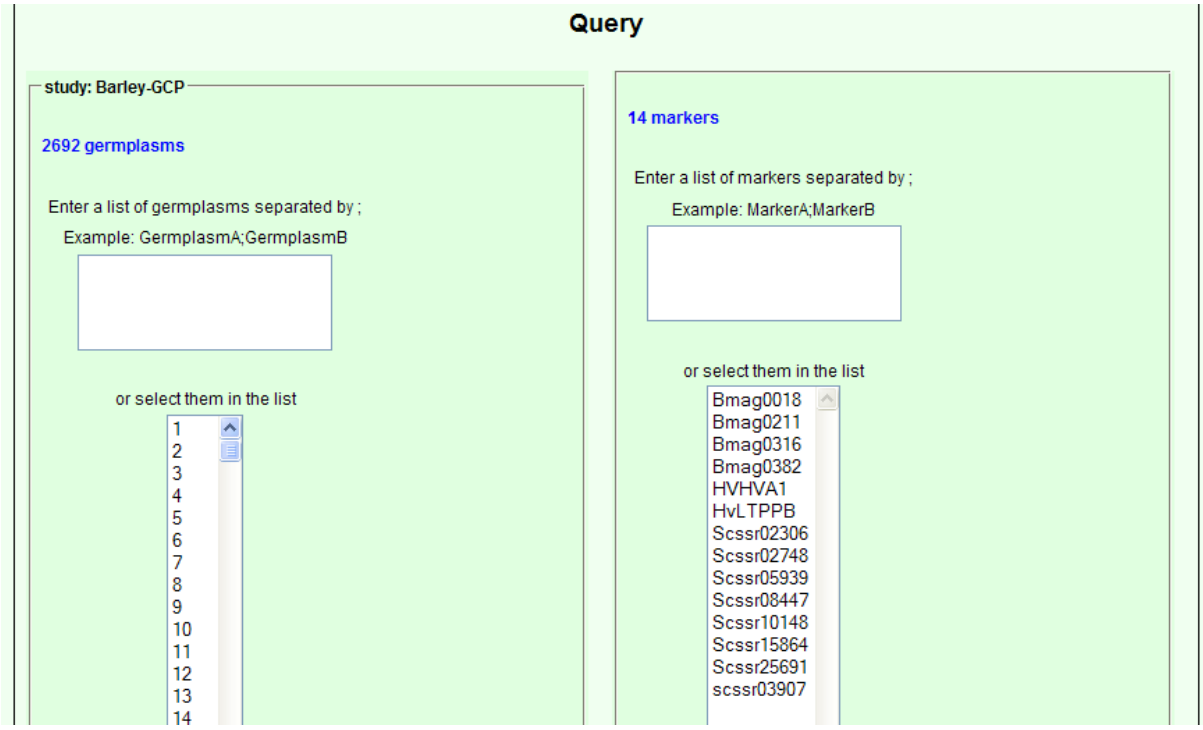
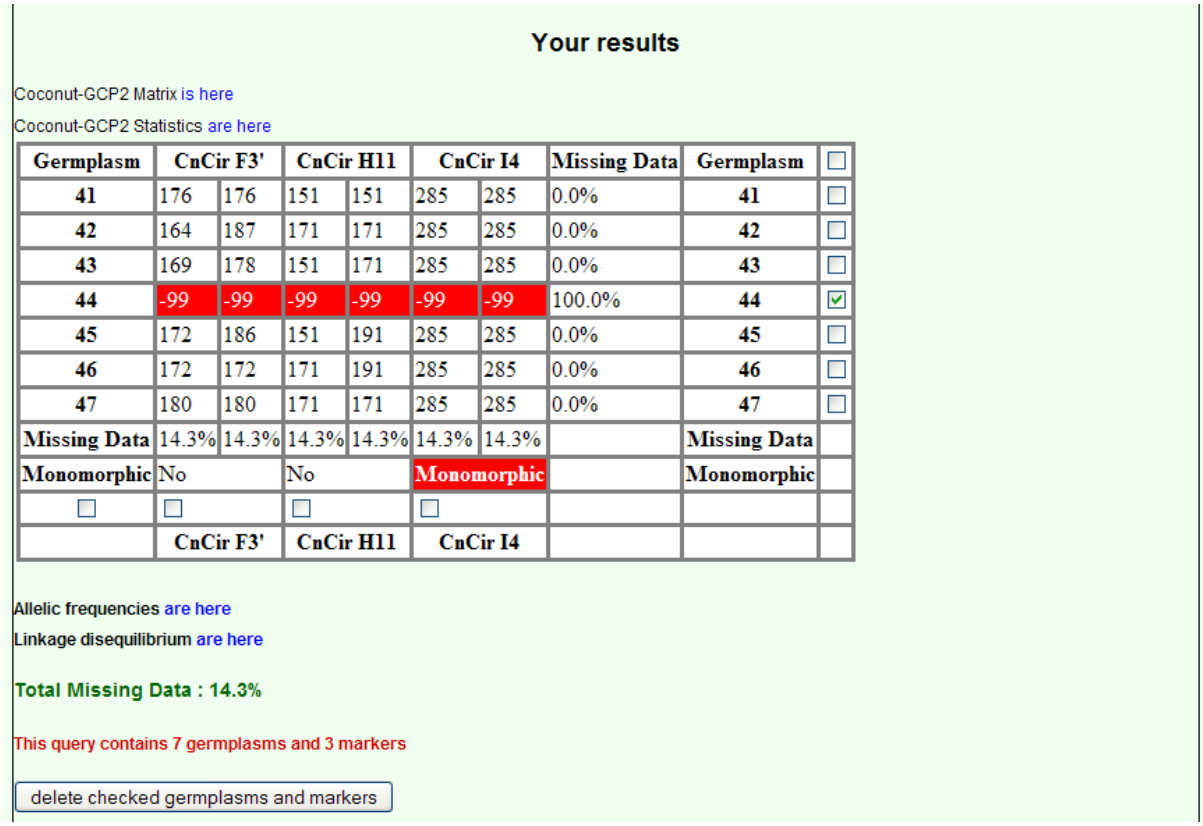


Figure 51 : Fenêtre de sélection des germplasmes et marqueurs associés à une étude de génotypage

L'utilisateur de GenDiversity peut sélectionner les germplasmes et les marqueurs qu'il souhaite pour renvoyer la diversité allélique pour chaque couple marqueur germplasme (Figure 52)



Your results

Coconut-GCP2 Matrix is here

Coconut-GCP2 Statistics are here

Germplasm	CnCir F3'		CnCir H11		CnCir I4		Missing Data	Germplasm	
41	176	176	151	151	285	285	0.0%	41	☐
42	164	187	171	171	285	285	0.0%	42	☐
43	169	178	151	171	285	285	0.0%	43	☐
44	-99	-99	-99	-99	-99	-99	100.0%	44	☑
45	172	186	151	191	285	285	0.0%	45	☐
46	172	172	171	191	285	285	0.0%	46	☐
47	180	180	171	171	285	285	0.0%	47	☐
Missing Data	14.3%	14.3%	14.3%	14.3%	14.3%	14.3%		Missing Data	
Monomorphic	No		No		Monomorphic			Monomorphic	
☐	☐		☐		☐				
	CnCir F3'		CnCir H11		CnCir I4				

Allelic frequencies are here

Linkage disequilibrium are here

Total Missing Data : 14.3%

This query contains 7 germplasms and 3 markers

delete checked germplasms and markers

Figure 52 : Matrice d'informations de diversité allélique

Fusion de résultats issus de plusieurs sources

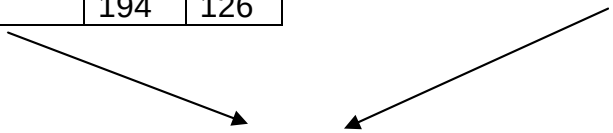
Gendiversity ne permet pas uniquement de visualiser des résultats de diversité allélique sous forme de matrice, il permet également de fusionner des matrices issues de sources différentes. Les données contenues dans ces sources permettent 3 types de fusions ayant des intérêts biologiques différents.

- La fusion de sources ayant des germplasmes et des marqueurs identiques permet de vérifier la cohérence de la diversité allélique. En effet, la découverte de mêmes allèles sur des analyses de diversités génétiques différentes apporte une vérification. Au contraire, la détection d'allèles différents remet en cause les allèles détectés et impliquera une nouvelle étude de diversité allélique pour confirmer.
- La fusion de sources ayant des germplasmes en commun mais utilisant des marqueurs génétiques différents (

- Figure **53**) permet d'obtenir une seule étude couplant plus de marqueurs. Cela permettra par exemple de faire des études plus fines sur l'histoire évolutive des germplasmes de cette étude.

Germplasmes/marqueurs	A	B
1	210	124
2	196	128
3	194	126

Germplasmes/marqueurs	C	D
1	184	168
2	188	176



Germplasmes/maqueurs	A	B	C	D
1	210	124	184	168
2	196	128	188	176

Figure 53 : Exemple de fusion de matrice pour des germplasmes communs

- La fusion de sources ayant des germplasmes différents mais des marqueurs génétiques identiques permet d'obtenir une étude unique possédant plus de germplasmes. Cela permettra de détecter les germplasmes ayant des allèles rares, ce qui pourrait par exemple être utile pour détecter le lien entre génotype et phénotype.

Difficultés à ajouter de nouvelles sources

GenDiversity permet d'interroger des données de génotypage directement dans leurs sources de données d'origine à l'aide d'adaptateurs dédié au médiateur Pantheon et plus précisément des adaptateurs de type Services Web développés sous BioMoby. Ces Services Web sont créés manuellement et nécessitent un travail d'implémentation important pour chaque source de données à intégrer. La difficulté et le temps nécessaire au développement de ces adaptateurs sont des goulots d'étranglement pour l'ajout de nouvelles sources de données dans GenDiversity. Nous avons donc décidé d'utiliser la plateforme BioSemantic pour développer de nouveaux Services Web Sémantiques permettant de réaliser le flux de travaux de GenDiversity mais également pour ajouter de nouvelles fonctionnalités à l'application.

7.6.2.1.Ajout de plusieurs modules TropGene

TropGene (Ruiz et al. 2004) est un schéma de base de données relationnel gérant les informations génétiques et génomiques de plantes tropicales étudiées au CIRAD. Ce schéma est utilisé dans différents modules correspondants chacun à une plante (e.g riz, sorgho,

bananier, etc.). Chaque module possède des variations au niveau du schéma mais tous sont basés sur le modèle TropGene. Il existe actuellement 9 modules différents de TropGene, et seulement 3 de ces modules sont intégrés dans GenDiversity. Nous avons donc choisi de tester notre approche sur le schéma relationnel de TropGene. Pour cela nous avons créé automatiquement une vue RDF du schéma de chaque module puis nous avons annoté manuellement des éléments de ce schéma à l'aide de concepts ontologiques issus du *GCP domain model*. Les concepts utilisés et l'élément qu'ils annotent sont :

- Le concept ontologique *GCP_Study* annote la colonne *name* de la table *study,*

- Le concept ontologique *GCP_Germplasm* annote la colonne *name* de la table *germplasm,*

- Le concept *GCP_GenomicFeatureDetector* annote la colonne *name* de la table *marker,*

- Le concept *GCP_GenomicFeatureVariant* annote la colonne *allele_type* de la table *genotypingdata*

Nous avons ensuite utilisé ces annotations pour créer automatiquement des Services Web Sémantiques correspondant aux différentes étapes du flux de travaux de GenDiversity. En sélectionnant un concept d'entrée et un concept de sortie nous pouvons directement créer les Services Web Sémantiques réalisant le même type d'action, pour tous les modules de TropGene. Ce flux de travaux nécessite la création de 4 SWS pour chaque module. Dans notre cas d'utilisation nous avons donc créé automatiquement ces 4 SWS pour chacun des 9 modules de TropGene. Nous avons donc créé automatiquement 36 SWS permettant d'utiliser GenDiversity sur l'ensemble des modules de TropGene, et ainsi de fusionner les données de diversité allélique de chacun de ces modules.

7.6.2.2.Ajout de nouvelles sources de données

MGIS[15] (Musa Germplasm Information System) est une base de données relationnelle contenant des informations sur les collections de bananiers (numéro d'accession, localisation géographique, pratiques agronomiques, conditions environnementales). La création de SWS interrogeant cette base de données permettrait d'avoir plus d'information sur les germplasmes utilisés dans une étude de génotypage. Nous avons donc créé et annoté une vue RDF du schéma de MGIS afin de créer des SWS récupérant la localisation géographique et les pratiques agronomiques pour un germplasme de bananier sélectionné dans GenDiversity.

[15] http://www.crop-diversity.org/banana/

Chado (C. J. Mungall et al. 2007) est un schéma de base de données relationnelle générique développée dans le cadre de GMOD (Generic Model Organism Database) (GMOD 2007). Le schéma de cette base de données relationnelle permet de représenter les types de données fréquemment utilisés en biologie comme par exemple une séquence, une comparaison de séquence, un phénotype ou encore un génotype. Il s'agit d'un des schémas relationnels les plus complexes actuellement disponible en biologie moléculaire. Le prix de cette généricité est la difficulté à prendre le schéma en main. Les nouveaux utilisateurs doivent passer beaucoup de temps pour devenir familier avec les fondamentaux du schéma. Chado est actuellement utilisé dans de nombreuses applications web dédiées à la génomique, notamment les modules plantes du projet GNPAnnot[16]. L'intégration du schéma de chado permettrait d'accéder à des données de séquences liées aux régions d'intérêts mises en évidence par GenDiversity

7.7. Perspectives

Indexer les concepts ontologiques référencés dans la plateforme

L'étape la plus consommatrice de temps dans notre plateforme est l'étape combinant la détection de vues RDF annotées avec les concepts ontologiques souhaités, la recherche d'un chemin reliant ces concepts, et la création de requête. Nous avons donc réfléchis à la manière la plus appropriée de diminuer le temps nécessaire à cette étape. Pour cela, nous souhaitons créer un index regroupant la totalité des concepts ontologiques utilisés dans les vues. Cela permettra au créateur de SWS de connaitre les concepts utilisés pour annoter les vues RDF ce qui lui donnera une aide à la sélection des concepts d'entrée et sortie qu'il souhaite utiliser. La 2ème fonctionnalité de cet index sera de relier les concepts ontologiques reliés entre eux sous la forme d'un triplet regroupant le nom de la vue puis les 2 concepts ontologiques reliés entre eux. Cette fonctionnalité optimisera considérablement la phase de recherche de plus court chemin en évitant les étapes préliminaires consistant à détecter les vues RDF annotées avec les concepts sélectionnés puis rechercher le chemin reliant ces concepts à chaque création de SWS. La création de cet index nécessite la mise en place d'une gestion de mise à jour. Cette mise à jour consistera à vérifier que les annotations présentes dans l'index et celles de la vue RDF sont bien synchronisées. Pour cela, il faudra, lors de la

[16] http://www.gnpannot.org/

modification d'une vue RDF, vérifier que les annotations de l'index sont toujours correctes et effectuer les mises à jour (ajout, modification, suppression).

Une fonctionnalité supplémentaire de cet index pourrait être de proposer la création automatique de workflow. Si 2 concepts A et B n'étant pas conjointement présents dans une vue RDF sont sélectionnés pour créer un SWS, cet index pourrait être utilisé pour proposer l'utilisation d'un concept ontologique intermédiaire C s'il existe un chemin reliant A et C dans une vue RDF et un autre chemin reliant B et C dans une autre vue RDF. Cela pourrait résulter en la création d'un SWS contenant 2 requêtes, la 2ème prenant en entrée le résultat de la première. Il n'est cependant pas certain qu'une telle approche puisse renvoyer des résultats cohérents.

Utiliser des standards d'échange de données bioinformatiques

Les bioontologies ont la particularité d'être composées d'un grand nombre de classes possédant très peu de propriétés. Des ontologies contenant des classes possédant plus de propriétés pourraient être utilisées pour annoter plus finement la vue RDF. Par exemple, si une classe Gene d'une ontologie possède des propriétés sur le type de données rattachées (nom, identifiant, synonymes, séquence, etc.), cette classe pourrait être utilisée pour annoter une table d'une base de données avec la classe Gene et les colonnes de cette table avec les propriétés associées à la classe. Cette classe pourrait alors être utilisée pour créer un Schéma XML définissant des types de données complexes utilisés par nos SWS. Ce type de classe n'existant pas, nous nous sommes limités à l'annotation d'une colonne d'une table avec une classe ontologique.

La standardisation des annotations SAWSDL a conduit à la création de schémas XML comportant des annotations SAWSDL vers des classes ontologiques ainsi que des propriétés vers des langages RDF (e.g DC, FOAF). C'est le cas du projet BioXSD dont l'objectif est la création d'un schéma XML se voulant être le format d'échange standard pour les données bioinformatiques et notamment pour la création de SWS bioinformatiques. L'utilisation du schéma XML de BioXSD pourrait permettre de réaliser des annotations plus fines sur nos vues RDF et ainsi autoriser la création de SWS consommant des types de données complexes. Ces types complexes permettraient la création de SWS ne renvoyant pas uniquement les données contenues dans une colonne d'une table mais renvoyant les données contenues dans plusieurs colonnes sous la forme d'un seul objet complexe.

7.8. Conclusion

Une plateforme de création automatisée d'adaptateurs sémantiques de type SWS nommée BioSemantic a été développée. Cette plateforme permet d'intégrer des bases de données relationnelles biologiques. Nous avons présenté l'architecture globale de BioSemantic ainsi que les étapes d'automatisations sur lesquelles nous avons concentré notre travail. Nos efforts se sont portés sur la création automatisée de vues RDF correspondant au schéma relationnel d'une base de données ainsi que sur l'utilisation de ces vues pour créer automatiquement des SWS. Les vues RDF, considérées comme des ontologies locales, sont créées automatiquement à l'aide de D2RQ. La création d'adaptateurs nécessite la détection de correspondances entre ces ontologies locales et des ontologies globales. Dans notre architecture ces ontologies globales sont des bioontologies de domaine. La seule étape manuelle actuellement dans notre architecture est la mise en correspondance entre ces ontologies globales et les ontologies locales. Elle est réalisée sous la forme d'annotations d'éléments de la vue RDF à l'aide de concepts issus de bioontologies.

Nous avons développé une application Web appelée BioSemantic Application qui s'appuie sur notre plateforme. Cette application Web offre des interfaces graphiques simples permettant de créer automatiquement une vue RDF puis de créer automatiquement des adaptateurs sémantiques sur plusieurs bases de données relationnelles en sélectionnant uniquement un concept ontologique d'entrée et un concept ontologique de sortie.

Dans cette partie du mémoire nous nous sommes focalisés sur la création des SWS. La création automatique de ces SWS nécessite la création automatique d'une requête pouvant interroger directement une base de données relationnelle à son emplacement d'origine. La partie de création automatique de ces requêtes est présentée en détail dans la prochaine partie du mémoire.

8. Création automatique de requête SPARQL

8.1. Introduction

La plateforme d'intégration de données BioSemantic s'appuie sur des mappings entre bases de données relationnelles et ontologies afin d'interroger des bases de données relationnelles réparties. Ces mappings sont utilisés pour créer et déployer des Services Web Sémantiques (SWS) de manière automatisée. Pour cela, nous nous sommes intéressés à la création automatique de requêtes prenant en entrée un type de données annoté avec un concept ontologique, et renvoyant un type de données également annoté avec un concept ontologique. Cette requête peut alors être intégrée automatiquement dans le SWS.

Les vues RDF créées dans la plateforme d'intégration de données sont structurées en RDF et représentent le schéma d'une source de données. Le RDF est basé sur la notion de triplets. La structure d'un triplet est de la forme sujet, prédicat, objet, où le sujet et l'objet peuvent être représentés par des nœuds et le prédicat peut être représenté par un arc orienté reliant le sujet à l'objet. L'ensemble des triplets d'un fichier RDF forme un graphe. Il est donc possible de parcourir une vue RDF à l'aide de parcours de graphes.

Dans la théorie des graphes, les problèmes de cheminement sont des problèmes classiques. L'objectif est de parcourir un chemin permettant de relier un nœud à un autre. Dans notre contexte, nous souhaitons pouvoir créer automatiquement une requête. Cette requête pourra alors être ajoutée automatiquement dans un SWS, rendant ainsi la création de SWS totalement automatique. Pour créer automatiquement des requêtes nous nous appuierons sur les vues RDF annotées, en y cherchant un plus court chemin pertinent reliant un concept ontologique d'entrée à un concept ontologique de sortie. Ce chemin unique permettra de créer une requête optimisant le temps d'interrogation ainsi que la pertinence des données renvoyées. La création de SWS à partir d'une vue RDF précédemment annotée sera ainsi totalement automatisée.

Dans une première partie nous allons présenter l'approche originale que nous avons développée pour détecter automatiquement un plus court chemin pertinent, permettant la création automatique d'une requête. Dans une seconde partie nous détaillerons les étapes permettant de passer de ce plus court chemin à une requête SPARQL. La troisième partie mettra en avant les résultats de notre approche en termes de pertinence de la requête mais

également en termes de vitesse d'exécution. Nous présenterons ensuite les perspectives avant de conclure.

8.2. Recherche de plus court chemin dans un graphe RDF

8.2.1. Contexte

8.2.1.1. Utilisation du plus court chemin

Dans la théorie des graphes le problème de plus court chemin consiste à trouver un chemin particulier entre deux nœuds d'un graphe. Dans un graphe pondéré orienté, le poids de chaque arc peut être défini. Il est alors possible de parcourir le graphe en tenant compte de l'orientation et du poids des arcs. Le coût d'un chemin correspond à la somme des coûts des arcs de ce chemin. Le plus court chemin est donc le chemin dont la somme des poids des arcs est le plus petit. Ce type d'algorithme est fréquemment utilisé pour trouver le plus court chemin permettant de se rendre d'un nœud à un autre d'un graphe non orienté de type carte géographique (Cook et al. 2008; Bellman 1958). Ces algorithmes peuvent également être utilisés pour répondre à des requêtes dans les bases de données de graphe (Angles & Gutierrez 2008). La recherche de plus court chemin est également utilisée dans le contexte des bases de données relationnelles où la recherche de chemin de jointures est important pour la recherche d'information sur plusieurs bases de données relationnelles (Domingos 2003). Hamill et al (Hamill & N. Martin 2004) décrivent une méthode permettant d'étendre les fonctionnalités d'une base de données relationnelle en autorisant l'interrogation sous forme de chemins basés sur des algorithmes de plus court chemin. Dans leur approche, le schéma relationnel d'une base de données est visualisé comme un graphe orienté dans lequel les arcs représentent des clés étrangères. Une requête prenant en entrée deux nœuds et un graphe est créée automatiquement. Cette approche ne tient pas compte des spécificités de parcours d'un graphe relationnel. Chaque relation parcourue est prise en compte de la même façon

8.2.1.2. Problème de combinatoire

Notre graphe peut être composé de plusieurs centaines de nœuds, et donc plusieurs centaines de liens orientés reliant ces nœuds, ce qui pose des problèmes pour rechercher rapidement le plus court chemin entre 2 nœuds. Par exemple, si un graphe possède 20 nœuds et que chaque nœud comporte un lien direct vers chaque autre nœud du graphe, le graphe

possède un total de 380 liens orientés. Dans ce cas, le nombre de chemins possibles pour relier 2 nœuds est supérieur à 6000 milliards. Cet exemple montre bien que des algorithmes évitant de calculer tous les chemins possibles sont nécessaires, de manière à éviter l'explosion combinatoire.

Il existe plusieurs algorithmes de recherche de plus court chemin. Chacun permet de trouver le plus court chemin reliant 2 nœuds d'un graphe. Ces algorithmes varient par leur complexité et par les caractéristiques possibles du graphe à parcourir.

8.2.1.3.Les différents algorithmes existants

L'algorithme de Bellman-Ford (Bellman 1958) autorise la présence de certains arcs de poids négatif et permet de détecter l'existence d'un circuit absorbant, c'est-à-dire de circuit de poids total négatif, accessible depuis le sommet source. Il est de complexité $O(n^3)$.

L'algorithme de Dijkstra (Dijkstra 1959) est un algorithme qui permet de déterminer le plus court chemin entre deux sommets d'un graphe connexe pondéré (orienté ou non) dont le poids lié aux arcs est positif ou nul. Il est de complexité $O(n^2)$.

L'algorithme de Floyd (Floyd 1962) est un algorithme qui permet de déterminer tous les poids des plus courts chemins d'un graphe orienté pondéré. Il est de complexité $O(n^3)$. Une étape supplémentaire est ensuite nécessaire pour récupérer tous les nœuds intermédiaires d'un chemin.

Dans notre cas d'utilisation, le passage d'une table à une autre table entrainera toujours l'augmentation de la taille du chemin. Il n'y a pas d'arc de poids négatif et donc pas non plus de circuit absorbant. C'est pour cela que nous avons décidé d'utiliser l'algorithme de Dijkstra comme base de notre algorithme de recherche de plus court chemin. Bien que les graphes RDF soient orientés, nous ne tiendrons pas compte de l'orientation d'un arc lors de notre étape de recherche de plus court chemin. Il sera donc possible parcourir le graphe en passant par le sens opposé d'un arc. L'orientation des arcs sera par contre utilisée dans la transformation du plus court chemin vers une requête (partie 8.3).

Dans la prochaine partie nous détaillerons l'algorithme de Dijkstra

8.2.1.4.L'algorithme de Dijkstra

L'algorithme de Dijkstra est basé sur le principe suivant :

Si le plus court chemin reliant E à S passe par les sommets s1, s2, ..., sk alors, les différentes étapes sont aussi les plus courts chemins reliant E aux différents sommets s1, s2, ..., sk.

On construit de proche en proche le chemin cherché en choisissant à chaque itération de l'algorithme, un sommet s_i du graphe parmi ceux qui n'ont pas encore été traités, tel que la longueur connue provisoirement du plus court chemin allant de E à s_i soit la plus courte possible.

Cet algorithme comporte 2 étapes. Une première étape qui consiste à initialiser les variables utilisées dans l'algorithme, puis une 2ème étape qui consiste à répéter des opérations tant que le sommet de sortie (S) n'est pas affecté d'un poids définitif.

```
Entrée
noeuds : liste des nœuds
début : nœud d'entrée
fin : nœud de sortie
Sortie
chemin : chemin le plus court permettant de relier début à fin
Paramètres
fils(n) : fonction permettant de renvoyer les fils d'un nœud n donné
distance(n1, n2) : fonction permettant de renvoyer la distance entre les nœuds n1 et n2
Fonction Dijkstra
    pour n parcourant nœuds
        n.parcouru = -1
        n.précédent = 0
    fin pour
    début.parcouru = 0
    pasEncoreVu = nœuds
    tant que (pasEncoreVu != liste vide)
        n1 = nœud dans pasEncoreVu avec parcouru le plus petit
        pasEncoreVu.enlever(n1)
        pour n2 parcourant fils(n1)
            si (n2.parcouru > n1.parcouru + distance(n1, n2))
                n2.parcouru = n1.parcouru + distance(n1, n2)
                n2.précédent = n1   // Dit que pour aller à n2, il faut passer par n1
            fin si
        fin pour
    fin tant que
    chemin = liste vide
    n = fin
    tant que n != début
        chemin.ajouterAvant(n)
        n = n.précédent
    fin tant que
    chemin.ajouterAvant(debut)
fin Fonction Dijkstra
```

Un applet Java montrant le déroulement des différentes étapes de l'algorithme de Dijkstra est disponible à l'adresse suivante:

http://www.dgp.toronto.edu/people/JamesStewart/270/9798s/Laffra/DijkstraApplet.html

Il s'agit d'une très bonne illustration, visualisée sous forme de graphe, des différentes étapes de cet algorithme.

8.2.1.5.Les limites de l'algorithme de Dijkstra

L'algorithme de Dijkstra peut être utilisé pour trouver le plus court chemin dans un graphe. Ce type d'algorithme permet de parcourir chaque nœud du graphe de la même façon. Il ne tient pas compte de la différence possible entre plusieurs types de nœud. Dans notre contexte de graphe correspondant à un schéma de base de données relationnelle, la simple recherche de plus court chemin peut s'avérer insuffisante. En effet, notre objectif final n'est pas de trouver le plus court chemin entre 2 nœuds d'un graphe, mais d'utiliser le plus court chemin pour trouver le chemin le plus pertinent permettant de créer une requête. Pour une question biologique donnée, nous considérerons un chemin comme pertinent s'il permet de créer automatiquement une requête renvoyant les mêmes données qu'une requête créée manuellement par un utilisateur expert de la BDR à interroger. Un chemin pertinent permettra donc de créer une requête pertinente.

Cette notion de requête pertinente combine plusieurs idées :

- éviter la détection de données redondantes,
- empêcher de renvoyer des résultats n'étant pas souhaités,
- éviter de ne pas renvoyer des résultats attendus.

Eviter les données redondantes

Le concept de normalisation dans les bases de données a été introduit par Edgar F. Codd en 1970 (E. F. Codd 1970) en définissant ce qu'on appelle actuellement la première Forme Normale. L'année suivante il définit la deuxième et la troisième Forme Normale (E.F Codd 1971). Une base de données est couramment considérée comme normalisée si elle est à la troisième Forme Normale (Date 1995). C'est cette normalisation qui permet d'éviter la redondance de données contenues dans la base. Dans notre cas, nous ne souhaitons pas vérifier la redondance des données enregistrées dans une base de données, mais éviter la redondance des données parcourues et renvoyées quel que soit son niveau de normalisation. Dans notre contexte, cette non-redondance consiste à ne renvoyer qu'une seule fois le même couple de données annotées avec les concepts ontologique d'entrée/sortie. Cela permettra d'améliorer les performances lors de l'exécution des requêtes.

C'est pour éviter la redondance des données renvoyées, que nous avons choisis de créer une requête en utilisant uniquement des jointures issues du plus court chemin reliant 2 nœuds. Ce plus court chemin sera transformé en requête en tenant compte des dépendances fonctionnelles au sein d'une même table mais également en tenant compte des dépendances d'inclusion lors du passage d'une table à une autre. Pour une question biologique donnée, nous souhaitons créer une requête renvoyant l'intégralité des données répondant à la question posée. La recherche du plus court chemin, pour créer cette requête, permet de limiter la redondance des données renvoyées et donc le temps d'interrogation par rapport à une requête créée en couplant tous les chemins reliant 2 nœuds d'un graphe. Par exemple, dans le schéma de la Figure 54, la requête créée pour répondre à la question biologique : renvoyer tous les germplasmes utilisés dans une étude de phénotypage donnée, sera créée en utilisant uniquement un des 2 chemins permettant de relier la table *Etude_phenotypage* à la table *Germplasme*. Ce chemin unique évitera de renvoyer des données redondantes. Dans notre exemple, il existe 2 chemins reliant les tables *Germplasme* et *Etude_phenotypage*. Le premier chemin passe par la table *Phenotype* et est représenté en rouge. Le deuxième passe par la table *Etude_phenotypage_germplasme* et est représenté en bleu. Les 2 chemins utilisent des dépendances d'inclusions relatives aux mêmes clés primaires (Etude_phenotypage.id_etude_phenotypage et Germplasme.id_germplasme). Le couplage de ces 2 chemins pour créer une requête peut donc potentiellement impliquer le renvoi de données redondantes.

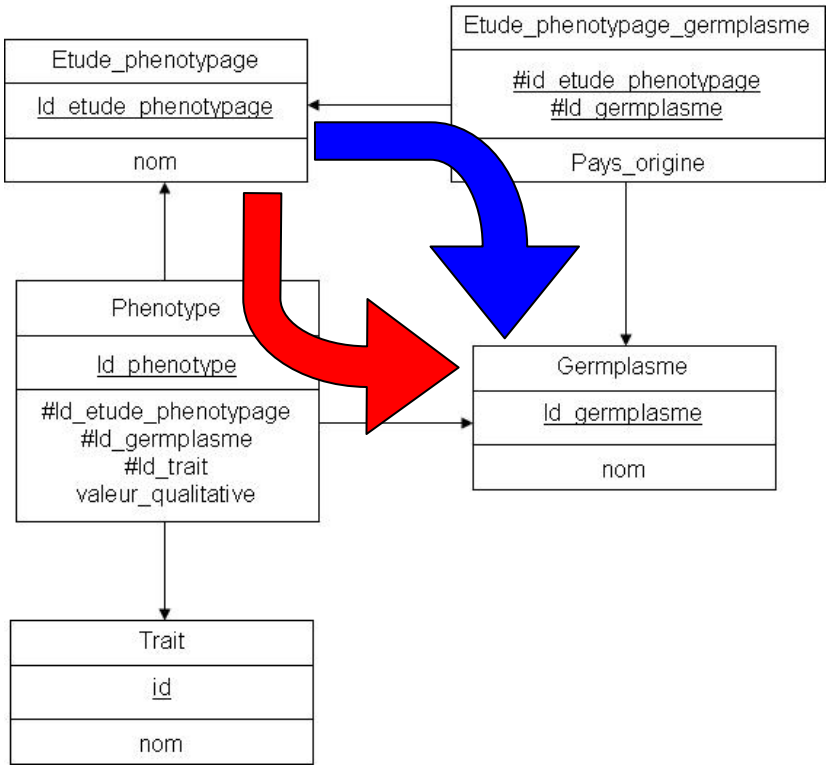


Figure 54 : Le plus court chemin pour éviter la redondance des données renvoyées

Empêcher le renvoi de résultats non souhaités

Dans notre approche nous posons l'hypothèse que l'utilisation d'un graphe où les nœuds représentent les tables d'un schéma de base de données et où les arcs représentent les dépendances d'inclusion entre 2 tables, empêche le renvoi de résultats non souhaités. Si un chemin est trouvé entre 2 nœuds d'un tel graphe, cela signifie que des dépendances d'inclusion permettent de lier les 2 tables correspondantes. Dans notre approche nous supposons que la présence d'une dépendance d'inclusion suffit à déterminer le lien entre 2 nœuds et donc le lien entre 2 tables, assurant ainsi la pertinence des données renvoyées.

En posant cette hypothèse nous sommes conscients de ne pas tenir compte de la sémantique d'une association.

Eviter de ne pas renvoyer des résultats souhaités

Dans notre approche nous posons l'hypothèse que chaque association entre 2 tables est un filtre diminuant le nombre de résultats possibles d'une requête. L'utilisation du plus court chemin nous permet donc de limiter un trop grand nombre de filtres évitant ainsi de ne pas renvoyer des résultats souhaités.

L'utilisation naïve d'un algorithme de détection de plus court chemin conduit à des requêtes ayant un grand pourcentage de résultats souhaités non renvoyés. Il faut tenir compte du contexte spécifique de la recherche de plus court chemin dans un schéma de base de données relationnelle. Une des spécificités de notre schéma relationnel par rapport à un simple graphe, est la présence de relations bien définies entre les tables. Notre recherche de plus court chemin doit tenir compte de ces types de relations entre les tables.

8.2.2. Prise en compte de la spécificité du modèle relationnel dans le parcours de graphe RDF

Lors de notre recherche de plus court chemin nous allons tenir compte des spécificités des relations parcourues. Pour cela, nous avons dans un premier temps répertorié les différentes relations utilisées lors de la conception de la base de données, et susceptibles d'augmenter la pertinence de nos requêtes créées automatiquement.

8.2.2.1.La combinaison de chemins

La relation d'agrégation: par défaut, une association exprime une relation à couplage faible. Les entités associées restent relativement indépendantes l'une de l'autre (Pierre-Alain Muller & Nathalie Gaertner 2000). L'agrégation est une forme particulière d'association qu'exprime un couplage plus fort entre entités. Elle permet d'exprimer des relations de type maître/esclaves et représentent des connexions bidirectionnelles dissymétriques. Mathématiquement, l'agrégation est une relation transitive, non symétrique et réflexive.

La relation de composition: il s'agit d'une forme d'agrégation avec couplage plus important entre les entités. Cette composition indique que la destruction de l'agrégat entraîne automatiquement la destruction des composants agrégés.

La relation d'héritage: la généralisation et la spécialisation sont des points de vue portés sur les hiérarchies d'entités. Une entité A est une spécialisation d'une entité B si chaque instance de A est une instance de B et si chaque instance de B est associée à au plus une instance de A.

Nous nous sommes intéressés aux relations d'agrégation et d'héritage car elles présentent une particularité commune susceptible de nous intéresser. Toutes les entités spécialisées, issues d'une même relation d'héritage, sont indépendantes. Cela signifie que l'intersection des instances qu'elle contient est vide. Cette propriété est également présente dans les entités agrégées issues d'un même agrégat.

Le fait que les entités agrégées issues d'une même agrégation et que les entités spécialisées issues d'une même relation d'héritage soient indépendantes**,** permet d'envisager que, dans le cas où notre algorithme parcours ce type de relation, une combinaison de chemins soit possible sans ajouter de redondance dans les données renvoyées. Chacun des chemins combinés renverrait alors des données indépendantes. L'indépendance des données renvoyées par chaque chemin combiné permet de créer une requête plus pertinente sans augmenter la redondance des données renvoyées. Le temps d'interrogation et la pertinence de ces requêtes sont donc optimisés.

Nous allons illustrer l'intérêt de la combinaison de chemins à l'aide de l'exemple présent dans le schéma d'entité-association de la Figure 55 ci dessous.

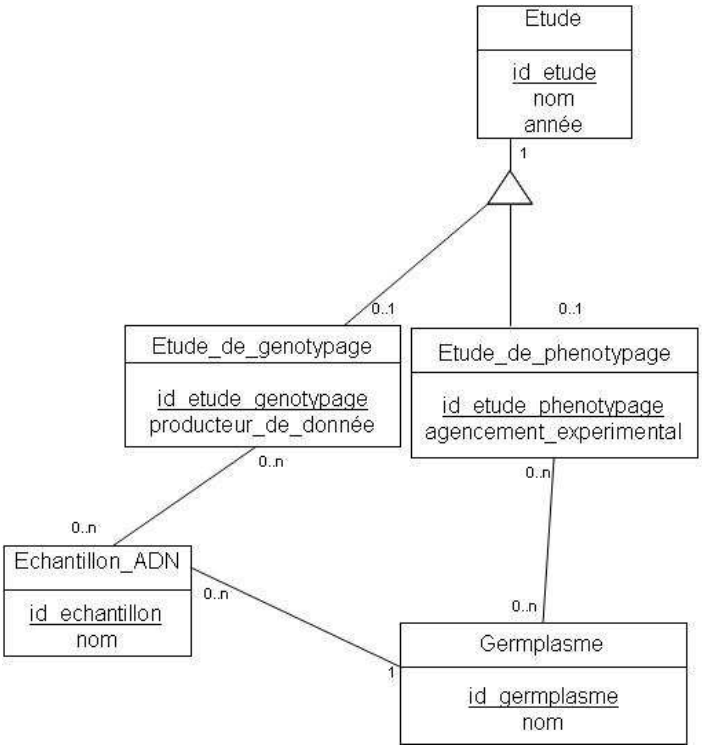


Figure 55 : Schéma entité-association montrant une relation d'héritage

Dans ce schéma on peut voir une relation d'héritage. Les entités *etude_de_genotypage* et *etude_de_phenotypage* sont des spécialisations d'une étude. Une étude est donc soit une étude de génotypage, soit une étude de phénotypage. Cela peut poser des problèmes si on souhaite créer automatiquement une requête qui, pour un identifiant d'étude donné, renvoie tous les noms de germplasmes associés. En utilisant un algorithme de recherche de plus court chemin, seul le chemin ayant le moins de nœuds intermédiaires permettant de relier *etude* à *germplasme* serait détecté. Dans notre exemple le chemin détecté serait le suivant:

etude – etude_de_phenotypage - germplasme

Ce chemin serait utilisé pour créer une requête composée de jointure entre ces tables. Cela signifie que si un identifiant donné correspond à un identifiant d'étude de génotypage, la requête créée automatiquement ne renverra aucune donnée. Dans le cas de relations d'héritage dans un schéma entité-association, il faut donc utiliser une combinaison de chemins pour la création de requêtes. Cependant, pour savoir comment prendre en compte ce type de relation, il faut s'intéresser aux règles de conversion de ce type de relation du modèle entité-association au modèle relationnel.

8.2.2.1.1.La conversion vers le modèle relationnel

La conversion d'une relation d'héritage peut s'effectuer de trois façons différentes lors du passage du modèle entité-association vers le modèle relationnel (Elmasri 2006).

- aplati vers le haut
- aplati vers le bas
- non aplati

Relation d'héritage aplati vers le haut: aplatir une relation d'héritage vers le haut consiste à regrouper tous les attributs des entités spécialisées dans l'entité généraliste. Dans notre exemple, cela revient à ne pas créer les relations *etude_de_genotypage* et *etude_de_phenotypage* mais à intégrer leurs attributs dans l'entité *étude.* Le modèle relationnelle résultant est le suivant:

- etude(id_etude, nom, annee, producteur_de_donnee, agencement_experimental, type_etude)
- echantillon_ADN(id_echantillon, nom, #id_germplasme)
- germplasme(id_germplasme, nom)
- echantillon_ADN_etude(#id_echantillon, #id_etude)
- etude_germplasme(#_id_germplasme, #id_etude)

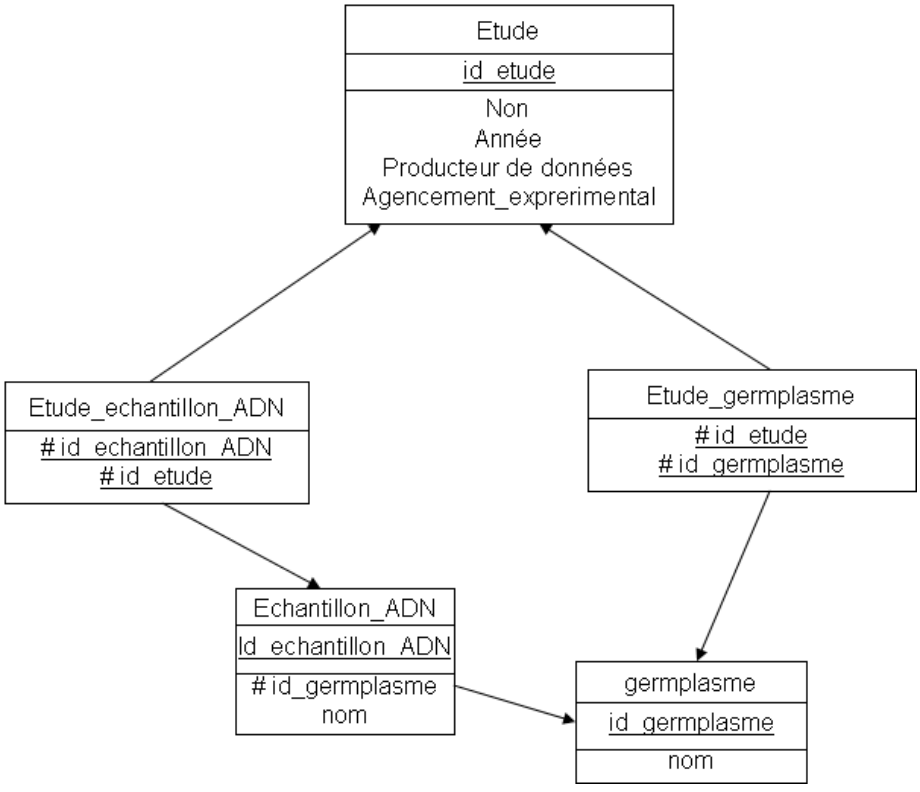


Figure 56 : Transformation du schéma entité association de la Figure 55 en schéma relationnel en aplatissant les relations d'héritage vers le haut

Ici les attributs soulignés correspondent à des clés primaires et les attributs précédés d'un dièse correspondent à des clés étrangères. Dans ce modèle relationnel, on voit bien que l'attribut *agencement_experimental* associé initialement à etude_de_genotypage, et l'attribut *producteur_de_donnee* associé initialement à *etude_de_phenotypage*, sont ici contenus dans la relation *etude*.

Le graphe RDF correspondant, créé automatiquement par D2RQ, est représenté dans la Figure 57. Par soucis de clarté, seules les tables (nœuds orange) sont représentées dans ce graphe RDF simplifié.

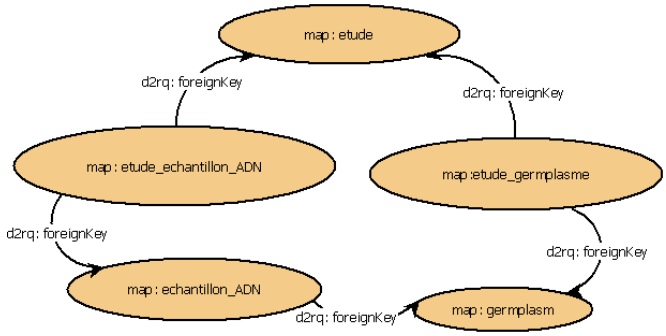


Figure 57 : Graphe RDF correspondant à la transformation du schéma entité association de la Figure 55 en schéma relationnel en aplatissant les relations d'héritage vers le haut

Relation d'héritage aplati vers le bas: aplatir une relation d'héritage vers le bas consiste à regrouper tous les attributs de l'entité généraliste dans les entités spécialisées. Dans notre exemple cela revient à ne pas créer la relation *etude* mais à intégrer ces attributs dans les relations *etude_de_genotypage* et *etude_de_phenotypage*. Le modèle résultant est le suivant:

- etude_de_genotypage (id_etude_genotypage, producteur_de_donne, nom, annee)
- etude_de_phenotypage (id_etude_phenotypage, agencement_experimental, nom, annee)
- echantillon_ADN(id_echantillon, nom, #id_germplasme)
- germplasme(id_germplasme, nom)
- echantillon_genotypage(#id_echantillon, #id_genotypingstudy)
- echantillon_phenotypage(#id_echantillon, #id_phenotypingstudy)

Dans ce modèle relationnel, la relation *étude* n'a pas été créée. Les attributs *nom* et *année*, initialement associés à l'entité *étude*, ont été dupliqués dans les relations *etude_de_genotypage* et *etude_de_phenotypage.*

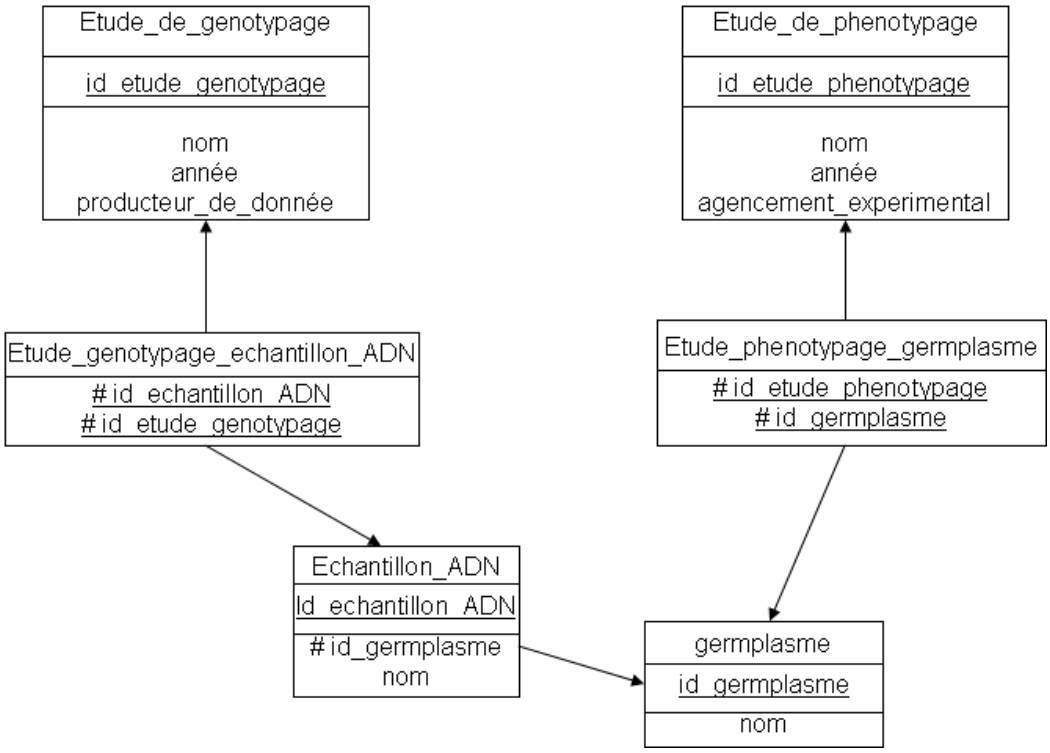


Figure 58 : Transformation du schéma entité association de la Figure 55 en schéma relationnel, en aplatissant les relations d'héritage vers le bas

Le graphe RDF correspondant, créé automatiquement par D2RQ, est représenté dans la Figure 59. Par soucis de clarté, seules les tables (nœuds orange) sont représentées dans ce graphe RDF simplifié.

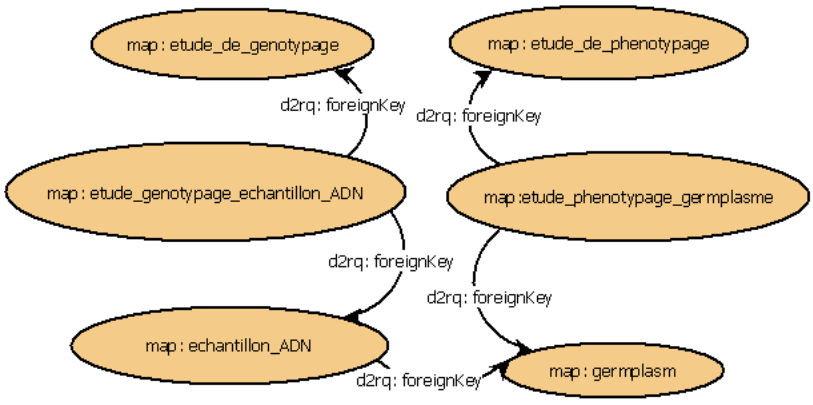


Figure 59 : Graphe RDF correspondant à la transformation du schéma entité association de la Figure 55 en schéma relationnel, en aplatissant les relations d'héritage vers le bas

Relation d'héritage non aplati: en suivant une approche non aplatie, chaque entité est convertie en une relation. Une clé étrangère supplémentaire, correspondant à la clé

primaire de la relation généraliste, est présente dans chaque relation spécialisée. Le modèle résultant est le suivant:

- etude(id_etude, nom, annee)
- echantillon_ADN(id_echantillon, nom, #id_germplasme)
- germplasme(id_germplasme, nom)
- etude_de_genotypage (id_etude_genotypage, producteur_de_donnee, #id_etude)
- etude_de_phenotypage(id_etude_phenotypage, agencement_experimental, #id_etude)
- echantillon_genotypage (#id_echantillon, # id_etude_genotypage)
- germplasme_phenotypage (# id_germplasme, # id_etude_phenotypage)

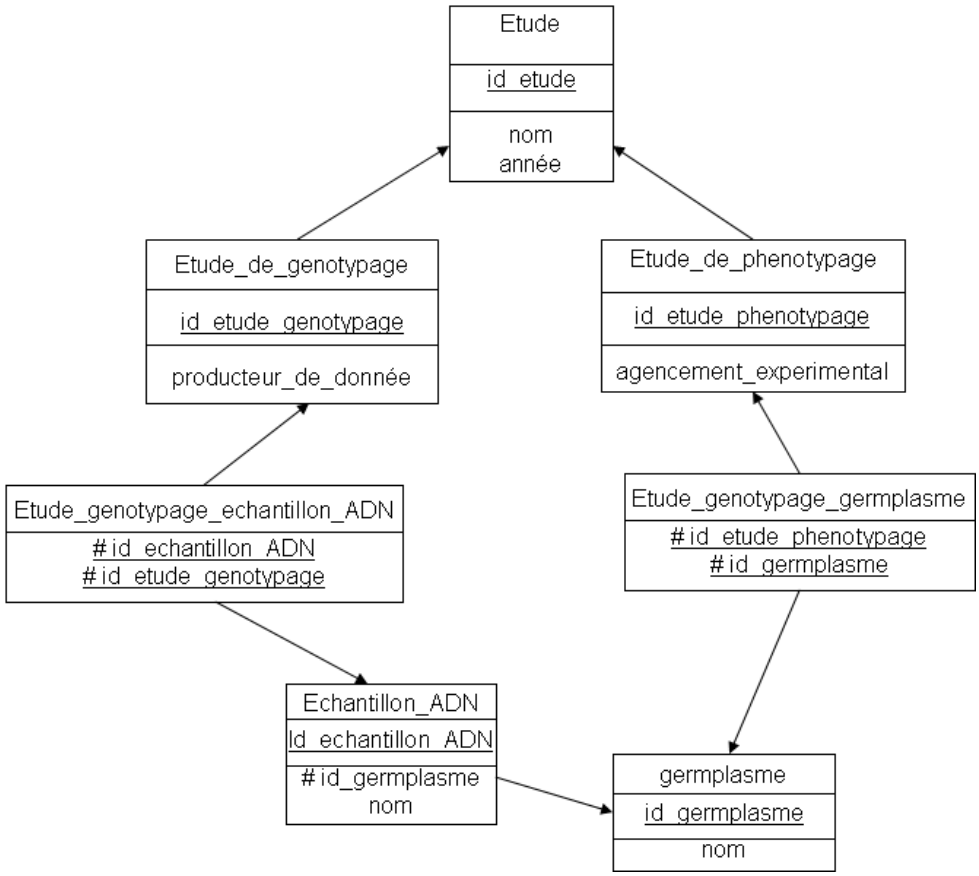


Figure 60 : Transformation du schéma entité association de la Figure 55 en schéma relationnel, en n'aplatissant pas les relations d'héritage

Dans ce modèle relationnel, les relations *étude*, *etude_de_genotypage* et *etude_de_phenotypage* sont présentent. De plus, la clé étrangère *id_etude* est ajoutée dans les relations *etude_de_genotypage* et *etude_de_phenotypage*. Cette clé étrangère correspond à la clé primaire de la relation *étude.*

Le graphe RDF correspondant, créé automatiquement par D2RQ, est représenté dans la. Par soucis de clarté, seules les tables (nœuds orange) sont représentées dans ce graphe RDF simplifié.

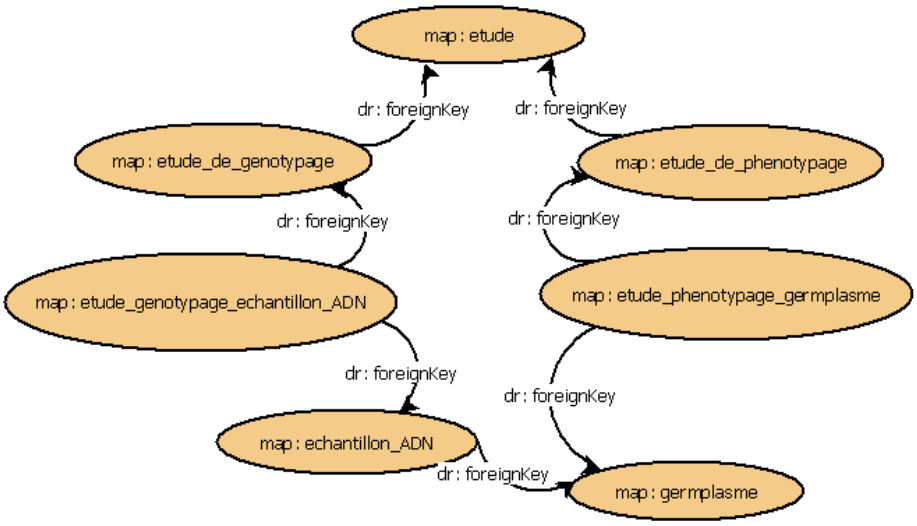


Figure 61 : Graphe RDF correspondant à la transformation du schéma entité association de la Figure 55 en schéma relationnel, en n'aplatissant pas les relations d'héritage

Nous souhaitons utiliser la recherche de plus court chemin entre deux nœuds d'un graphe RDF dans le but de créer automatiquement une requête. L'utilisation simple de l'algorithme de Dijkstra implique la création d'une requête ne renvoyant pas tous les résultats attendus. La prise en compte des relations d'héritage pourrait résoudre en partie ce problème. Nous avons détecté les 3 façons de formaliser une relation d'héritage dans un modèle relationnel. Il faut maintenant savoir si toutes ces formes posent des problèmes de renvoi de données.

8.2.2.1.2.Quelles formes d'héritage prendre en compte ?

L'héritage **aplati vers le haut** induit l'absence physique de relations d'héritage. Les informations de spécialisation sont rattachées à la classe générique. Toutes les données sont réunies dans la même table, ce qui signifie pour nous qu'elles sont rattachées au même nœud du graphe. Si toutes les données sont rattachées au même nœud, il n'y a pas besoin de parcourir d'autres nœuds pour récupérer l'intégralité de l'information. Ce type de conversion de relation d'héritage n'a donc pas à être détecté dans notre approche de recherche de plus court chemin.

Il sera par contre nécessaire d'annoter la vue RDF avec des concepts ontologiques en utilisant les annotations conditionnelles proposées par D2RQ. Cela permettra de pouvoir créer une requête en interrogeant les entités spécialisées. Dans ce cas, une entité généraliste sera annotée avec plusieurs concepts ontologiques. Un concept sera utilisé pour annoter l'entité généraliste et les autres concepts seront utilisés, sous une condition définie explicitement à l'aide d'une balise d2rq:condition du vocabulaire D2RQ, pour annoter chaque entité spécialisée.

L'annotation nécessaire à la prise en compte de ce genre de relations d'héritage ne peut pas être réalisée automatiquement. Il est donc nécessaire de réaliser cette étape lors de l'annotation sémantique manuelle de la vue RDF de manière à prendre en compte la relation d'héritage lors de la création automatique de requête, sans avoir à modifier l'algorithme de détection de plus court chemin.

L'héritage **aplati vers le bas** induit également l'absence physique de relations d'héritage. Les informations de la relation généraliste sont dupliquées dans les différentes relations spécialisées. Il n'est donc pas possible de détecter automatiquement ces relations d'héritage en parcourant la vue RDF. Il est cependant nécessaire de prendre en compte ce genre de relation d'héritage car, les informations généralistes étant dupliquées dans plusieurs relations, le renvoi de la totalité des instances associées à cette relation va nécessiter de combiner plusieurs chemins.

La solution envisagée pour prendre en compte les relations d'héritages aplaties vers le bas est l'annotation de chaque relation spécialisée avec le même concept ontologique. La prise en compte de ce type de relation d'héritage nécessite donc une étape manuelle d'annotation de la vue RDF

Lors d'une conversion vers une relation d'héritage **non aplati**, les relations spécialisées et généralistes sont présentes physiquement. Cela signifie qu'une relation est créée pour chaque entité. Dans ces conditions, la création d'une requête prenant en entrée la relation généraliste peut nécessiter de combiner plusieurs chemins. Pour la conversion non aplatie de la relation d'héritage, présentée dans la Figure 60, la création d'une requête renvoyant tous les germplasmes pour une étude donnée, nécessitera la combinaison des plus courts chemins reliant les 2 relations et passant par les relations *etude_de_genotypage* et *etude_de_phenotypage*.

8.2.2.1.3.La détection des relations d'héritage non aplatis

La détection automatique de relations d'héritage non aplati pour la transformation d'un schéma relationnel vers une ontologie a été décrite dans (Tirmizi et al. 2008). Elle est également utilisée pour typer les relations d'héritage de l'outil DB2OWL (Cullot et al. 2007)**.** Cette détection automatique est basée sur les techniques d'ingénierie inverse dans les bases de données, tentant notamment de convertir un modèle relationnel en modèle entité association (K. H. Davis & Arora 1987). Cette détection utilise la particularité des contraintes d'intégrité entre les tables généralistes et les tables spécialisées. En effet, pour qu'une table soit une spécialisation d'une autre table, elle doit contenir pour seule clé étrangère la clé primaire de la table généraliste.

La détection automatique de ce genre de relation d'héritage est donc rendu possible en utilisant la règle suivante:

Subclass(r,s) <- Rel(r)^Rel(s)^PK(x,r)^FK(x,r,_,s)
avec
Rel(r)r est une relation
PK(x,r)x est la clé primaire de r
FK(x,r,y,s) x est la clé primaire de la relation r et référence y dans la relation s

L'utilisation de cette règle rend automatique la détection de toutes les relations d'héritage non aplati. Elle ne permet par contre pas de faire la distinction entre une relation d'héritage et les relations d'agrégation ou de composition. Ce qui signifie qu'en utilisant cette règle nous détecterons également toutes les relations d'agrégation et de composition. Or nous souhaitons également combiner les chemins lors du parcours de relations d'agrégation. L'utilisation de cette règle nous permet donc de détecter tous les types de chemins à combiner.

8.2.2.1.4.L'annotation de la vue RDF

Dans la partie précédente nous avons déterminé une règle permettant de détecter les relations d'agrégation et de composition, ainsi que certaines relations d'héritage.

Il faut maintenant s'intéresser à la façon dont ces relations vont être décrites de manière à en tenir compte dans la création automatique de requêtes.

Nous avons décidé d'ajouter cette information directement dans la vue RDF, à l'aide de la propriété *rdfs:SubClassOf* issue du vocabulaire RDF. Dans notre exemple, la détection de relation d'héritage implique la création de deux nouveaux triplets dans la vue RDF.

etude_de_genotypage rdfs:subClassOf etude

etude_de_phenotypage rdfs:subClassOf etude

8.2.2.2.Prise en compte de pondérations dans la sélection de chemins

Dans notre approche nous avons posé l'hypothèse que le chemin le plus court, entre deux nœuds d'une vue RDF, serait le chemin le plus à même d'être utilisé pour créer automatiquement une requête. Dans un schéma relationnel, différents types d'associations existent. La simple prise en compte du nombre de nœuds à parcourir pour déterminer la longueur du plus court chemin ne prend pas en compte cette diversité d'associations.

Nous allons donc déterminer si certaines associations peuvent favoriser la création d'une requête cohérente. Si c'est le cas, il serait intéressant de pondérer le chemin le plus court en donnant un poids faible au parcours d'associations favorisant la création de requête pertinente et en donnant un poids élevé au parcours d'associations limitant la création de requêtes pertinentes.

Dans notre vue RDF nous n'avons pas accès aux cardinalités des associations. Nous ne pouvons donc pas utiliser ce critère pour favoriser le passage par une association plutôt que par une autre. Nous avons par contre accès aux clés primaires et clés étrangères. Nous allons donc utiliser ces informations pour tenter d'optimiser l'algorithme de détection de plus court chemin. Nous allons dans un premier temps nous intéresser aux tables d'associations puis aux arités des relations.

8.2.2.2.1.Critères de pondérations

Table d'association:

Lors du passage du schéma entité association au schéma relationnel, une entité devient une relation contenant des attributs. Les associations binaires, de type 1:1 ou 1:n disparaissent au profit d'une clé étrangère dans la relation coté 0,1 ou 1,1. Cette clé étrangère référence la clé primaire de l'autre relation.

Une association binaire de type n:m, est convertit en une relation qui, dans le modèle physique, correspondra à une table d'association. La clé primaire de cette table est alors composée de 2 clés étrangères référençant les 2 clés primaires des 2 tables associées. Dans notre contexte de recherche de plus court chemin reliant 2 nœuds d'un graphe, il serait peut être intéressant de ne pas traiter ce type de table de la même façon qu'une table représentant une entité

Arité d'une table d'association:

Pour chaque nœud traversé dans notre graphe, nous utilisons une clé étrangère pour arriver à un nœud spécifique. Nous utilisons également une seule clé étrangère pour passer à un autre nœud. Une table d'association comporte les instances représentant l'intersection des instances comprises dans les 2 tables associées. Cependant, une table d'association n'a pas forcément une arité de 2. Elle peut associer plus de 2 tables. Dans le cas d'une table d'association n-aire, c'est à dire associée à n tables, elle contiendra les instances correspondant à l'intersection des instances pour les n tables associées. L'association binaire, est schématisée dans la partie gauche de la Figure 62. La partie verte représente les instances contenues dans la table d'association. Ces instances correspondent à l'intersection des instances contenues dans les tables associées. Si, lors de la recherche d'un plus court chemin, nous traversons une telle table d'association, nous sommes donc certains de prendre en compte toutes les instances associant un marker à un germplasm. La partie droite de la Figure 62 schématise quand à elle une association ternaire. La zone verte représente les instances contenues dans la table d'association résultante. Dans notre recherche de plus court chemin, la prise en compte des tables d'association conduit à un chemin linéaire. Cela signifie que, dans le chemin résultant, le nœud représentant la table d'association ne sera associé qu'à 2 autres tables. La prise en compte d'une 3ème association dans la table d'association ne sera donc pas utile dans notre chemin et, au contraire, conduira à une perte d'information présentée en rouge dans la partie inférieure de la Figure 62.

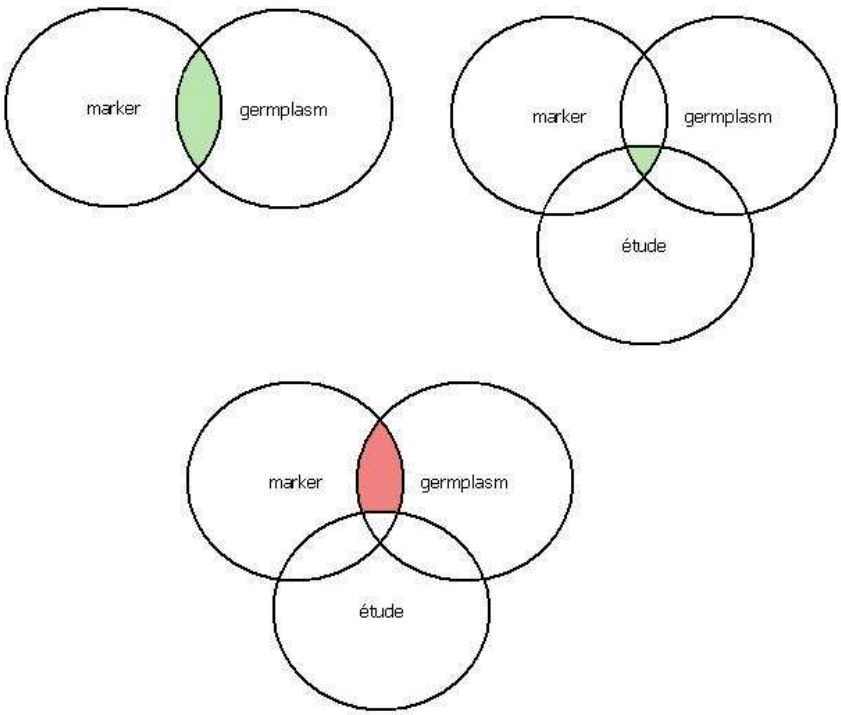


Figure 62 : Représentations des données contenues dans 2 tables d'association d'arité différente

Le passage par des tables d'association d'arité supérieur à 2 peut induire une perte d'information. Nous avons donc intérêt à favoriser le passage par les tables d'association binaires mais également de pénaliser le passage par les tables d'association d'arité supérieure à 2.

Choix de l'annotateur de la vue RDF :

Lors de l'interrogation d'une base de données relationnelle, le temps d'exécution de la requête augmente si la requête parcours une table contenant une très grande quantité de données. L'expert annotant manuellement la vue RDF connait le modèle relationnel de la base de données ainsi que les données qu'elle contient. S'il sait qu'une table comporte un nombre de tuples suffisant pour induire un temps de requêtage important, il peut vouloir empêcher le passage par cette table.

8.2.2.2.2.Détection des éléments du modèle relationnel dans la vue RDF

Table d'association

Une table d'association peut être représentée de 2 façons différentes dans la vue RDF. Si la table d'association possède des attributs, elle est représentée comme les autres tables par la propriété *d2rq:Classmap*. Dans ce cas il est nécessaire de différencier les tables d'associations des autres tables. La Figure 63 présente l'algorithme de détection des tables d'association avec attributs.

pk= primary key of R
fk=foreign keys of R
$if((\forall u \in R)(u \in fk \Rightarrow u \in pk))\{$
$if((\forall u \in R)(u \in pk \Rightarrow u \in fk))\{$
R is an association table
}
}

Figure 63 : Algorithme de détection des tables d'association avec attributs

Table d'association sans attributs :

Une table d'association sans attributs est un cas particulier de table d'association. Il s'agit d'une table associant 2 autres tables et dont les seules colonnes sont celles correspondant aux clés primaires des 2 tables associées. Dans la Figure 64**,** c'est le cas de la table d'association *germplasme_phenotypage* créée suite au passage du schéma entité association de la Figure 55 vers un modèle relationnel sans aplatir les relations d'héritage. Ce type de table

d'association est représenté par une propriété *d2rq:propertyBridge* et non pas par une *d2rq:classmap* comme c'est le cas pour les autres tables d'association. La détection de ces tables d'association sans attributs se fait automatiquement en détectant les propriétés *d2rq:propertybridge* de la vue RDF possédant plusieurs clés étrangères.

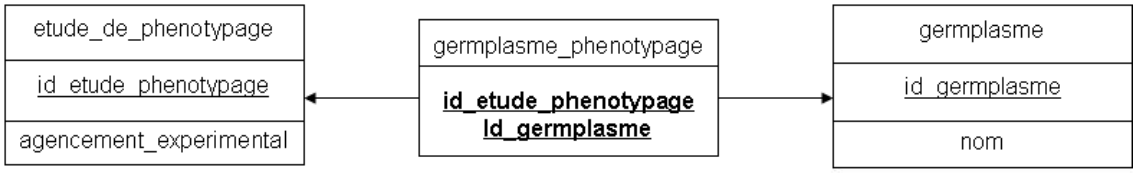


Figure 64 : Exemple de table d'association sans attributs

Arité :

Dans la vue RDF, l'information sur l'arité d'une table d'association est transcrite indirectement. Cette arité peut être trouvée automatiquement en détectant le nombre de clés étrangères présentent dans une table d'association.

8.2.2.2.3.Annotation des éléments du modèle relationnel dans la vue RDF

Les annotations présentées dans cette partie sont ajoutées automatiquement à la vue RDF lors de sa création.

Tables d'association :

Les tables d'associations possédant ou non des attributs sont annotées avec la propriété *dr:associatedTo*. Un triplet contenant ce prédicat sera ajouté pour chaque table associée. Le sujet de ce triplet correspondra à la table d'association et l'objet à la table associée.

Arité :

L'arité d'une table d'association est annotée avec la propriété *dr:arity*. Une propriété *dr:arity* peut également être ajoutée manuellement par un annotateur souhaitant éviter le passage par une table précise.

8.2.2.2.4.Exemples d'annotations

Nous avons présentés les différents typages automatiques dont nous avons besoin pour tenir compte du contexte particulier de passage par une table d'association lors de la recherche du plus court chemin reliant 2 nœuds d'un graphe. Ce typage est réalisé automatiquement, sous la forme de triplets, lors de la création de la vue RDF.

Dans les exemples de la Figure 64, les triplets suivants sont ajoutés à la table d'association binaire nommée *markergermplasm* associant la table *marker* et la table *germplasm*:

map:germplasme_phenotypage dr:associatedTo map:germplasme
map:germplasme_phenotypage dr:associatedTo map:etude_de_phenotypage
map:germplasme_phenotypagedr:arity"2"^^rdf:int

Les triplets suivant seront ajoutés pour la table d'association ternaire nommée *markergermplasmetude* associant les tables *marker*, *étude* et *germplasme*

map:markergermplasmetudedr:associatedTomap:germplasme
map:markergermplasmetudedr:associatedTomap:etude
map:markergermplasmetudedr:associatedTomap:marker
map:markergermplasmetudedr:arity"3"^^rdf:int

8.2.3. Prise en compte des spécificités du modèle relationnel dans l'algorithme

Nous avons montré que l'utilisation d'un algorithme de recherche de plus court chemin ne prenant pas en compte le contexte spécifique de notre approche ne permet pas de créer automatiquement une requête cohérente. Nous avons donc défini des conditions supplémentaires que nous souhaitons utiliser dans notre algorithme de recherche de plus court chemin sous conditions.

8.2.3.1.Le type de relations et le type de nœuds

Les informations utiles pour le parcours de graphe puis pour la transformation en requête sont les suivantes :

- le sujet
- l'objet
- la longueur du chemin parcouru entre le nœud de départ et le pointeur
- le type de nœud
- le type de relation

Le sujet: il s'agit du nœud sur lequel se trouve le pointeur. La valeur de cet attribut correspond à l'URI déterminant la table sur laquelle on se situe. Un sujet ne peut être qu'un nœud de type table. Si la valeur du sujet est NULL, cela signifie que toutes les informations sur la table courante ont été récupérées et que l'algorithme détecte le type d'association.

L'objet: il s'agit du nœud pour lequel le pointeur cherche une information. La valeur de cet attribut correspond à l'URI de l'élément correspondant. Cet élément peut être une table ou une colonne.

Le type de nœud: cet attribut correspond au type de nœud de l'objet. Différents types de nœuds sont parcourus. Chaque type de nœud induit un parcours de graphe différent. Un type de nœud prend une des valeurs suivantes:

Concept d'entrée: c'est la valeur du type de nœud pour le pointeur initial. Le premier nœud que l'on parcourt est le nœud annoté avec le concept ontologique d'entrée. Ce nœud est ajouté automatiquement au début de l'algorithme.

Concept de sortie: c'est la valeur du type de nœud pour le pointeur final. Il s'agit d'un pointeur ajouté automatiquement lorsque l'on se situe sur le nœud annoté avec le concept ontologique de sortie.

Clé primaire: valeur prise lorsque le nœud vers lequel on pointe correspond à une colonne utilisée pour définir la clé primaire d'une table

Simple colonne: valeur prise lorsque le nœud vers lequel on pointe ne correspond à aucun autre

Clé étrangère: valeur prise lorsque le nœud vers lequel on pointe correspond à une clé étrangère

Clé étrangère inverse: les arcs liant les nœuds de notre graphe sont des arcs orientés. Une clé étrangère inverse correspond au passage par un arc décrivant une clé étrangère, mais en parcourant dans l'orientation opposée à celle de l'arc.

Une table: le nœud vers lequel on pointe ne correspond pas forcément à une colonne, il peut également correspondre à une table. Il s'agit de la valeur prise lorsque le nœud vers lequel on pointe est une autre table que celle sur laquelle on se situe.

Le type d'association: à chaque nœud parcouru est associé un type d'association. Il s'agit de la propriété qui permet de lier le sujet à l'objet. Cette propriété peut prendre une des valeurs suivantes:

Table d'association: valeur prise lorsque l'on passe d'une table à une autre et que la table définie par l'objet est une table d'association.

Table d'association sans attributs: valeur prise lorsque l'on passe d'une table à une autre et que la table définie par l'objet est une table d'association sans attributs. Il faut

différencier ce cas particulier des autres tables d'association car il n'est pas géré de la même façon lors de la transformation du plus court chemin en requête.

Relation d'héritage: valeur prise lorsque l'on passe d'une table à une autre et que la table définie par l'objet est une spécialisation de la table définie dans le sujet.

Relation d'association: valeur prise lorsque l'on passe d'une table à une autre sans parcourir une relation d'héritage ou d'agrégation et sans pointer vers une table d'association

Pas une association: valeur prise lorsque l'on passe d'une table vers une colonne.

La longueur du chemin: Sa valeur correspond à la longueur du chemin entre le concept d'entrée et le nœud courant. Cette valeur de longueur du chemin augmente uniquement lorsque l'on passe d'une table à une autre. Le cout d'un arc permettant de passer d'une table à une autre varie suivant le type de relation.

Plus le coût de l'arc est faible, plus le chemin sera favorisé. L'ordre de grandeur de l'incrémentation du chemin est le suivant :

table d'association binaire < association binaire=relation d'héritage < table d'association n-aire < association n-aire

Il est possible de modifier facilement la valeur associée à chaque type d'arc. Les valeurs utilisées par défaut sont les suivantes :

- table d'association binaire : 0,98
- association binaire : 1
- relation d'héritage : 1
- table d'association n-aire : 0,98 + arité
- association n-aire : 0,985 + arité

8.2.3.2.Implémentation des spécificités du modèle relationnel dans l'algorithme

Dans cette partie nous allons présenter l'algorithme utilisé pour trouver le plus court chemin reliant 2 nœuds dans un graphe. Cet algorithme est présenté sous forme de pseudo code:

```
Entrée
  vueRDF: vue RDF contenant tous les triplets du schém de la base de données
  noeudEntree: noeud correspondant à l'annotation de départ de l'algorithme
  noeudSortie : noeud correspondant à l'annotation d'arrivée de l'algorithme
Sortie
  chemin : plus court chemin permettant de relier l'entrée à la sortie
```

```
Paramètres
  colonneEntree : colonne correspondant à l'annotation d'entrée
  tableEntree : table contenant l'annotation d'entrée
  colonneSortie: colonne correspondant à l'annotation de sortie
  tableSortie: table contenant l'annotation de sortie
  longueur : longueur du chemin. Cette valeur est incrémentée au cours de l'algorithme et est initialisée à 0
  vecteurTables    : vecteur comportant toutes les tables de la vue RDF. Chaque table y possède un nom et un
                     poids. Le poids d'une table est initialisé à -1
  table : table utilisée à un instant t de l'algorithme
  heritage: paramètre permettant de définir qu'une table ajoutée au chemin provient d'une relation d'héritage
  tableAssoc: paramètre permettant de définir qu'une table ajoutée au chemin provient d'une relation
                     d'héritage
  cleEtr: paramètre permettant de définir qu'une colonne ajoutée au chemin est une clé étrangère
  cleEtrInv          : paramètre permettant de définir qu'une colonne ajoutée au chemin est une clé étrangère
                     inverse. C'est à dire reliée à une autre table en parcourant un arc du graphe correspondant à une
                     clé étrangère mais dans le sens inverse
  clePrim: paramètre permettant de définir qu'une colonne ajoutée au chemin est une clé primaire
  valeur: poids accordé à une table d'association
  trouveChemin: booléen prenant la valeur true si une ligne est créée dans le chemin
Fonction recherchePlusCourtChemin
//ajoute le noeud d'entrée au chemin et lui associe une longueur de chemin (ici 0)
chemin.ajoute(noeudEntree, longueur)
chemin.ajoute(colonneEntree, longueur)
tant que (vecteurTables n'est pas vide)
  //récupère la table de vecteurTables ayant le plus court chemin
  table = vecteurTables.plusCourtChemin
  //la longueur courante prend la valeur de la table de plus court chemin
  longueur = table.longueur
  si (table !=tableSortie)
    //supprime la table courante du vecteur de tables
    vecteurTables.supprime(table)
  fin si
  //ajoute la clé primaire de la table courante dans le chemin
  chemin.ajoute(table.clePrimaire, longueur, clePrim)
  si (table = tableEntree)
    chemin.ajoute(tableEntree, longueur)
  fin si
  sinon
    //si on accede à cette table en traversant une relation d'héritage
    si (table.estHeritage)
      chemin.ajoute(table,longueur,heritage)
    fin si
    //si la table courante est une table d'association
    sinon si(table.estTableAssoc)
      chemin.ajoute(table, longueur, tableAssoc)
    fin sinon si
    sinon
      chemin.ajoute(table, longueur)
    fin sinon
  fin sinon
  si (table = tableSortie)
    chemin.ajoute(colonneSortie, longueur)
    chemin.ajoute(noeudSortie, longueur)
  fin si
  //si la table courante possède une ou plusieurs clés étrangères
  si(table.aCleEtrangere)
    //cleEtrangere est une clé étrangère reliant la table courante à une autre table
    pour chaque cleEtrangere
      //tableEtrangere est une table reliée par une clé étrangère à la table courante
      tableEtrangere = table.relie
```

```
    //vérifie que la tableEtrangere n'a pas encore été parcourue ou que la longueur de chemin actuelle est plus
    //court que celle de cette table
    si (tableEtrangere.longueur == -1 ou tableEtrangere.longueur > longueur)
      chemin.ajoute(cleEtrangere, longueur, cleEtr)
      si(tableEtrangere.estHeritage)
        chemin.ajoute(tableEtrangere, longueur+1, heritage)
      fin si
      sinon si(tableEtrangere.estTableAssoc)
        //la variable arite prend la valeur correspondant au nombre de clé étrangère de la table
        //d'association divisé par 100
        arite = tableEtrangere.nombreCleEtrangeres
        chemin.ajoute(tableEtrangere, longueur+valeur+arité, tableAssoc)
      fin sinon si
      sinon
        chemin.ajoute(tableEtrangere, longueur)
      fin sinon
      //modifie la valeur de la longueur de chemin des tables reliées à la table courante par des clés
      //étrangères
      tableVecteur.modifie(tableEtrangere)
    fin si
  fin pour chaque
 fin si
 //si la table courante possède une ou plusieurs clés étrangères inverses. C'est à dire si elle est reliée à une autre
 //table en parcourant un arc correspondant à une clé étrangère mais dans le sens inverse
 si(table.aCleEtrangereInverse)
   //cleEtrangereInverse est une clé étrangère inverse reliant la table courante à une autre table
   pour chaque cleEtrangereInverse
     //tableEtrangereInverse est une table reliée par une clé étrangère inverse à la table courante
     tableEtrangereInverse = table.relie
     chemin.ajoute(cleEtrangereInverse, longueur, cleEtrInv)
     si(tableEtrangereInverse.estHeritage)
       chemin.ajoute(tableEtrangereInverse, longueur+1, heritage)
     fin si
     sinon si(tableEtrangereInverse.estTableAssoc)
       //la variable arite prend la valeur correspondant au nombre de clé étrangère de la table d'association divisé
       //par 100
       arite = tableEtrangereInverse.nombreCleEtrangeres
       chemin.ajoute(tableEtrangereInverse, longueur+valeur+arité, tableAssoc)
     fin sinon si
     sinon
       chemin.ajoute(tableEtrangereInverse, longueur)
     fin sinon
     //modifie la valeur de la longueur de chemin des tables reliées à la table courante par des clés étrangères
     //inverses
     tableVecteur.modifie(tableEtrangereInverse)
   fin pour chaque
 fin si
 si(table = tableSortie)
   //vide le vecteur de tables si la table courante correspond à la table de sortie. De ce fait l'algorithme prend fin
   vecteurTables.vide
 fin si
fin Fonction rechercheePlusCourtChemin
```

8.3. Création de requêtes SPARQL

Dans la partie précédente nous avons déterminé les modifications que nous devions apporter à un algorithme classique de recherche de plus court chemin reliant 2 nœuds d'un

graphe. Ces modifications nous permettent de détecter un plus court chemin utilisable pour créer automatiquement une requête. La requête créée automatiquement devra être compatible avec la plateforme d'intégration de données mise en place. Cette plateforme est basée sur des SWS intégrant des requêtes SPARQL. Nous allons présenter dans cette partie la méthodologie utilisée pour transformer notre plus court chemin en requête SPARQL. Cette étape de transformation prend en compte l'orientation des arcs du graphe RDF.

Pour illustrer cette étape de création de requête nous allons nous appuyer sur l'exemple d'une base de données comportant 7 tables. Le modèle relationnel de cette base de données est représenté dans la Figure 65 ci dessous. Dans cette figure, les clés primaires sont soulignées et les clés étrangère sont précédées du caractère #.

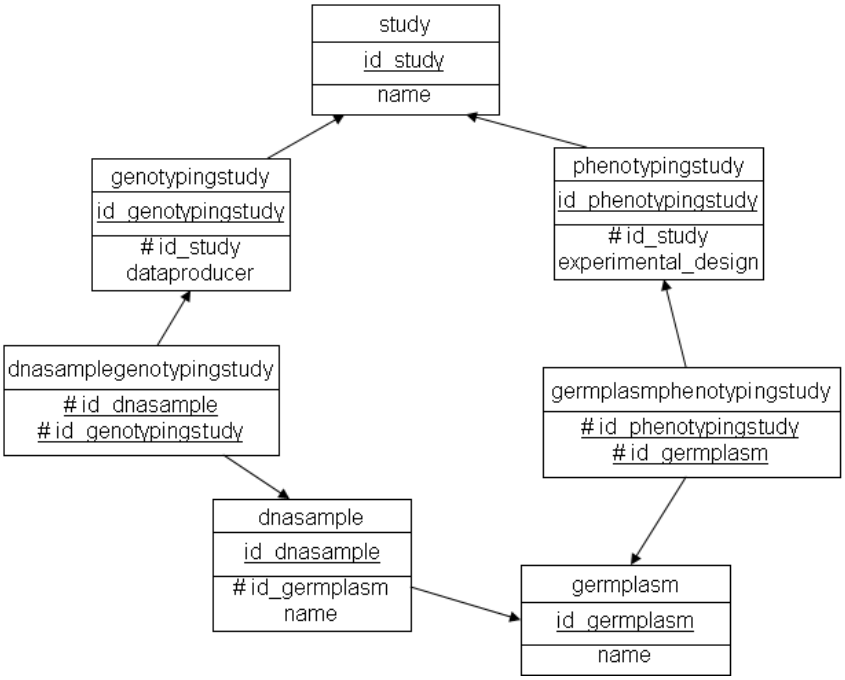


Figure 65 : Modèle relationnel de la base de données utilisée comme exemple de la création automatique de requête SPARQL

Ce schéma a conduit à la création automatique d'une vue RDF dont la représentation sous forme de graphe est présentée dans la Figure 66. Dans ce graphe, seul les nœuds représentant des tables sont présents, ces nœuds sont de couleur orange. Les nœuds rouges représentent les annotations sémantiques, ajoutées manuellement, réalisées sur une colonne de la table associée. Les arcs noirs représentent les propriétés présentent d'origine dans la vue RDF, et les arcs rouges représentent les arcs rajoutés automatiquement par notre approche. Les nœuds bleus représentent la valeur associée à l'arité d'une table d'association, qui est détectée automatiquement.

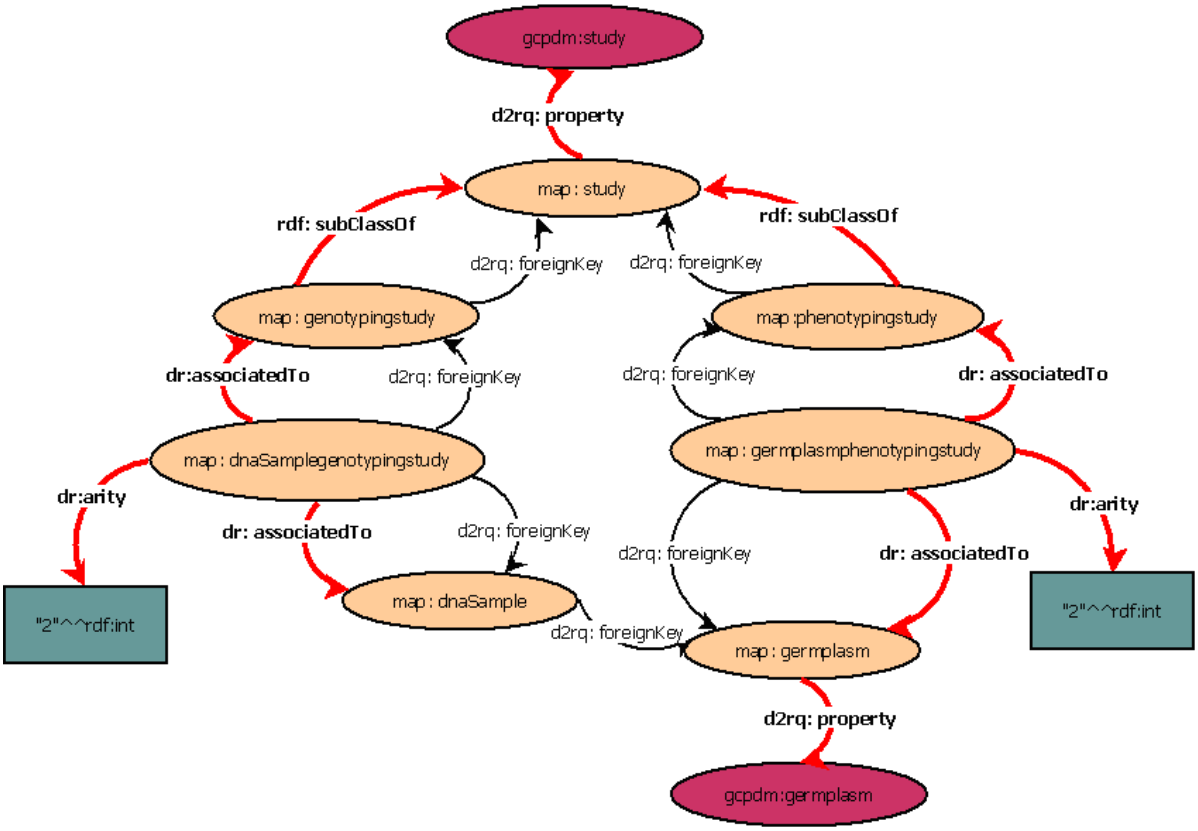


Figure 66 : Graphe représentant la vue RDF du schéma de la base de données utilisée comme exemple de la création de requête SPARQL

8.3.1. Sélection des concepts d'entrée et de sortie de la requête SPARQL

Pour créer une requête SPARQL, nous utilisons les annotations sémantiques ajoutées manuellement dans la vue RDF d'un schéma de base de données relationnelle. Chaque annotation sémantique ajoutée à une vue RDF peut être potentiellement utilisée comme entrée ou sortie d'un SWS. Dans l'exemple de la Figure 66, nous allons sélectionner les annotations sémantiques *gcpdm:study* et *gcpdm:germplasm*.

8.3.2. Détails du fonctionnement de l'algorithme

Les concepts ontologiques sont utilisés pour trouver le plus court chemin reliant l'entrée à la sortie pour une vue RDF donnée. Comme décrit précédemment, ce plus court chemin peut être la combinaison de plusieurs plus courts chemins dans le cas ou des relations d'héritage sont parcourues.

Dans notre exemple nous souhaitons répondre à la question biologique : renvoyer tous les germplasmes utilisés dans une étude biologique donnée. Pour répondre à cette question nous allons détecter le plus court chemin sous condition permettant de relier la donnée d'entrée, annotée ici avec le concept *gcpdm:study*, à la donnée de sortie, annoté ici avec le concept *gcpdm:germplasm*. Dans cette étape de recherche de plus court chemin, l'orientation des arcs n'est pas prise en compte. Dans notre exemple nous devons parcourir une relation de spécialisation entre la table *study* et les tables *genotypingstudy* et *phenotypingstudy*. Le plus court chemin détecté sera donc l'agrégation des plus courts chemins passant par ces 2 tables. Les plus courts chemins détectés sont représentés dans la Figure 67. Dans cette figure, les chemins utilisés pour créer la future requête sont représentés à l'aide de lignes bleues.

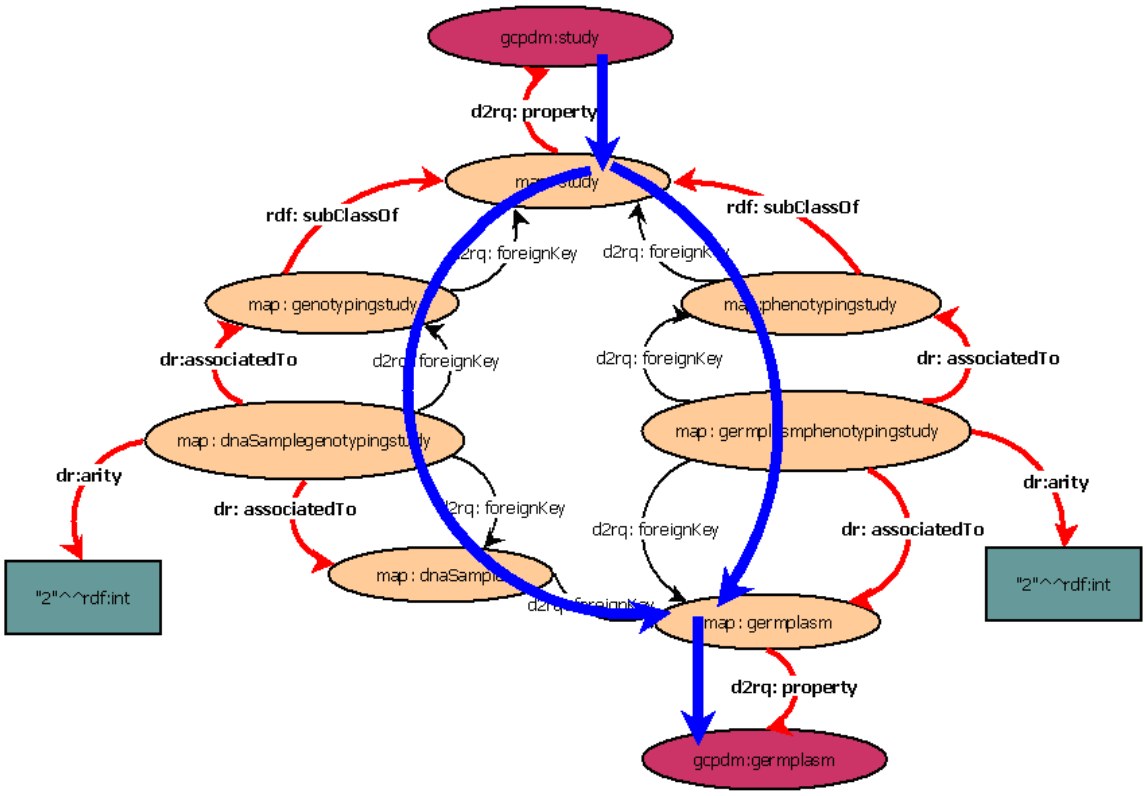


Figure 67 : Plus court chemin sous condition détecté pour relier les concepts ontologiques *gcpdm:study* et *gcpdm:germplasm*

Au niveau de l'implémentation, le résultat de la recherche de plus court chemin est un vecteur. Chaque ligne de ce vecteur comporte des informations sur un nœud parcouru. Le vecteur résultant de la recherche de plus court chemin pour notre exemple est présenté dans l'annexe B.

8.3.2.1.Création automatique des triplets de la requête SPARQL

A ce stade de la création de requêtes nous avons à notre disposition la vue RDF d'un schéma relationnel, les concepts ontologiques d'entrée et sortie ainsi qu'un plus court chemin permettant de relier l'entrée à la sortie. Nous disposons donc de tous les éléments nécessaires à la création d'une requête SPARQL.

La première étape de transformation de notre plus court chemin consiste à créer la structure de triplets de la requête finale.

Chaque triplet de la requête SPARQL que nous allons créer est structurée de la façon suivante:

sujet	**prédicat**	**objet**
clé primaire	annotation	colonne correspondante

Dans notre plus court chemin, nous allons utiliser les informations sur le type de nœud pour savoir s'il s'agit d'une table ou d'une colonne. Dans le cas d'une colonne, le fait qu'il s'agisse d'une clé primaire ou non permet de définir le sujet ou l'objet d'un triplet alors que l'annotation associée à la colonne sera utilisée comme prédicat du triplet. Le type d'association que l'on parcourt nous permettra de déterminer de qu'elle manière il faut traiter les données du prochain nœud parcouru.

Le pseudo code correspondant à la transformation du plus court chemin en triplets est présenté ci dessous:

```
Entrée
  plusCourtChemin : plus court chemin reliant 2 noeuds du graphe
Sortie
  requeteObtenue : première version de la requête structurée sous forme de triplets
Paramètres
  simpleColonne : variable pour les associations de type simple colonne
  clePrim : variable pour les associations de type clé primaire
  cleEtr : variable pour les associations de type clé étrangère
  cleEtrInv : variable pour les associations de type clé étrangère inverse
  tablesAssoSansAttributs[] : tableau listant toutes les tables d'associations sans attributs et les attributs de leur
clé primaire
  tableAssociationSansAttributs : booléen prenant la valeur true quand une table d'association sans attributs est
parcourue
  noeudCourant : noeud du plus court chemin qui est en train d'être parcouru
  typeAssociation : type d'association du noeud courant (heritage, table d'association, association)
  typeNoeud : type du noeud courant (colonne, clé primaire, clé étrangère, clé étrangère inverse, table)
  longueur : longueur du chemin pour le dernier noeud parcouru (initialisé à -1)
Fontion creationDeTriplets
  //pour chaque ligne du tableau plusCourtChemin
  pour i de 0 à plusCourtChemin.taille{
    //le noeud courant prend la valeur de la ligne i du tableau plusCourtChemin
    noeudCourant = plusCourtChemin[i]
    // récupère le type d'association du noeud courant
    typeAssociation = noeudCourant.typeAsso
    si (longueur < noeudCourant.longueur)
      si (noeudCourant.sujet == NULL)
```

```
  si (noeudCourant.typeNoeud = colonne)
    //clePrim prend la valeur du sujet du noeud courant
    clePrim = noeudCourant.objet
  fin si
  sinon si(noeudCourant.typeNoeud == clé primaire)
    clePrim= noeudCourant.sujet
    si (clé primaire composée de plusieurs attributs)
      tant que(noeudCourant.typeNoeud == clé primaire)
        i++
      fin tant que
      //récupère l'URI précédant le nom de la ressource
      espaceNom = noeudCourant.espaceNom
      //récupère le nom local de la ressource, c'est à dire le nom présent derrière l'URI
      nomLocal= plusCourtChemin[i+1].nomLocal
      clePrim=espaceNom+"key_"+nomLocal
    fin si
    sinon
      //ajoute une ligne de triplets à la requête en prenant clePrim comme prédicat
      requeteObtenue.ajoute(triplets, clePrim)
    fin sinon
  fin sinon
fin si
sinon
  si(noeudCourant.typeNoeud == clé étrangère)
    cleEtr = noeudCourant.objet
      si( i+4 < plusCourtChemin.longueur)
        si (clé primaire composée de plusieurs attributs)
          si ( tableAssociationSansAttributs == true)
            tableAssociationSansAttributs = false
            si ( plusCourtChemin[i+2].longueur < longueur)
              requeteObtenue.ajoute(triplets, simpleColonne)
            fin si
            i++
          fin si
          sinon
            requeteObtenue.ajoute(triplets, cleEtr)
            j=i+2;
            tant que(plusCourtChemin[j].typeNoeud = clé primaire)
              clePrim=plusCourtChemin[j].objet
              j++
            fin tant que
          fin sinon
        fin si
        sinon
          requeteObtenue.ajoute(triplets, cleEtr)
          clePrim = plusCourtChemin[i+2].objet
        fin sinon
      fin si
    fin si
    si (noeudCourant.typeNoeud == colonne)
      simpleColonne = noeudCourant.objet
      si (plusCourtChemin[i-2].typeNoeud = clé primaire)
        requeteObtenue.ajoute(triplets, simpleColonne)
      fin si
      sinon
        requeteObtenue.ajoute(triplets, clePrim)
      fin sinon
    fin si
    si (noeudCourant.typeNoeud == clé étrangère inverse){
      cleEtrInv = noeudCourant.objet
```

```
        si (i+3 > plusCourtChemin.longueur)
          requeteObtenue.ajoute(triplets, cleEtrInv)
        fin si
        sinon si(plusCourtChemin[i+1] a une clé primaire composée d'un seul attribut)
          requeteObtenue.ajoute(triplets, annotationSortie)
        fin sinon si
        sinon
          clePrim = simpleColonne
        fin sinon
      fin si
    fin sinon
  fin si
  sinon
    //parcours tous les elements de la table tableAssociationSansAttributs contenant toutes les tables
    //d'assocation sans attributs
    pour (j de 0 à tablesAssoSansAttributs.taille)
      tableCourante = tablesAssoSansAttributs[j]
      si(tableCourante = noeudCourant.objet)
        si( tableCourante.attribut1 == clé pointant vers la dernière table pacourue)
          clePrim = tableCourante.attribut2
          requeteObtenue.ajoute(triplets, tableCourante)
        fin si
        sinon si( tableCourante.attribut1 == clé pointant vers la prochaine table à parcourir)
          clePrim = tableCourante.attribut1
          requeteObtenue.ajoute(triplets, tableCourante)
        fin sinon si
      fin si
    fin pour
    si (plusCourtChemin[i+1] == concept de sortie de la requête)
      si(plusCourtChemin[i+2] == null || plusCourtChemin[i+2].longueur == 0)
        requeteObtenue.ajoute(triplets, annotationSortie)
      fin si
    fin si
  fin sinon
 fin si
fin pour
fin Fonction creationDeTriplets
```

L'exemple de schéma de base de données utilisé comme fil conducteur de ce chapitre conduit à la création d'un plus court chemin long et difficile à lire facilement. Ce plus court chemin ainsi que les triplets créés en l'utilisant sont tout de même présentés dans l'Annexe B du mémoire. Pour illustrer clairement l'algorithme de passage du plus court chemin vers les triplets de la future requête SPARQL, nous allons nous appuyer sur un schéma de base de données trivial présenté dans la Figure 68. Une vue RDF de ce schéma va ensuite être créée automatiquement. Sa représentation sous forme de graphe RDF est présentée dans la Figure 69. Dans ce graphe RDF la base de données est représentée par un nœud vert, les tables par des nœuds orange et les colonnes par des nœuds gris. Ce graphe RDF a ensuite été annoté manuellement à l'aide de 2 concepts ontologiques issus de l'ontologie de modèle de données GCP (nœuds rouge de la Figure 69). Ces annotations vont être utilisées comme entrée et sortie

de la requête correspondant à la question biologique suivante : quels sont les marqueurs utilisés dans une étude donnée.

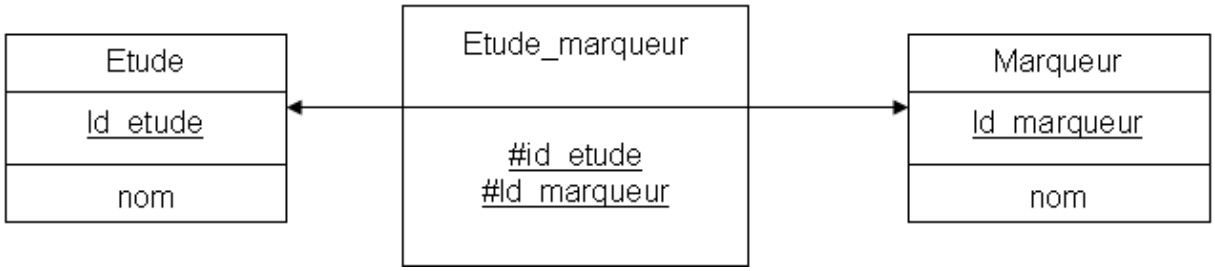


Figure 68 : Schéma relationnel trivial utilisé pour illustrer clairement le passage du plus court chemin vers des triplets de la requête SPARQL

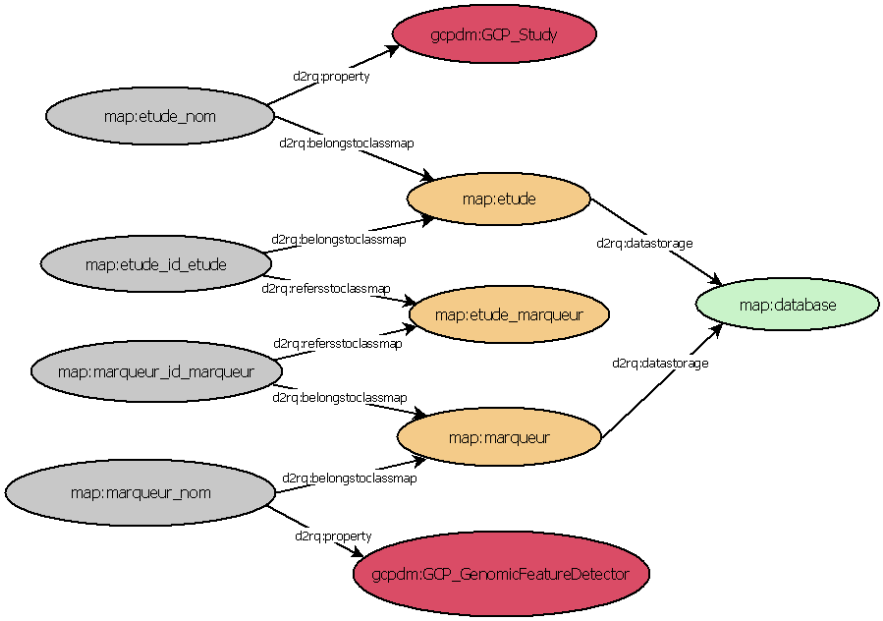


Figure 69 : Graphe RDF annoté correspondant au schéma de la Figure 68

Le plus court chemin détecté par notre algorithme est représenté dans la Figure 70. On peut noter que la longueur du chemin n'est incrémentée que lors du passage par un nœud de type table. Le passage par une table d'association binaire incrémente la longueur du chemin de 0,98 alors que le passage par une relation d'association binaire l'augmente de 1.

Les triplets créés pour ce chemin sont représentés dans la Figure 71. Dans notre exemple 3 triplets sont créés. Chaque triplet possède un sujet qui est une variable, un prédicat qui permet de définir la localisation dans le graphe RDF et un objet qui est également une variable. Le premier triplet permet de récupérer les identifiants correspondant à chaque nom d'étude. Le deuxième triplet permet de récupérer l'identifiant de marqueur correspondant à un

identifiant d'étude donné. Enfin, le dernier triplet permettra de récupérer le nom correspondant à un identifiant de marqueur donné.

Sujet	Objet	longueur chemin	type de noeud	type association
null	gcpdm:GCP_Study	0.0	entrée	-
null	map:etude_nom	0.0	colonne	-
null	map:etude_id_etude	0.0	clé primaire	-
map:etude	map:etude_marqueur_id_etude	0.0	clé étrangère	-
map:etude	map:etude_marqueur	0.98	table	table d'association
null	map:etude_marqueur_id_marqueur	0.98	clé primaire	-
null	map:etude_marqueur_id_etude	0.98	clé primaire	-
map:etude_marqueur	map:etude_marqueur_id_marqueur	0.98	colonne	-
map:etude_marqueur	map:marqueur_id_marqueur	0.98	clé étr. inv.	-
map:etude_marqueur	map:marqueur	1.98	table	relation d'association
null	map:marqueur_nom	1.98	colonne	-
null	gcpdm:GCP_GenomicFeatureDetector	1.98	sortie	-

Figure 70 : Plus court chemin

```
?etude_id_etude http://medoc.cirad.fr#etude_nom ?etude_nom.
?marqueur_id_marqueur http://medoc.cirad.fr#etude_marqueur ?etude_id_etude.
?marqueur_id_marqueur http://medoc.cirad.fr#marqueur_nom ?marqueur_nom
```

Figure 71 : Triplets

8.3.3. Traitement du cas particulier des relations d'héritage dans le graphe RDF

La prise en compte des relations d'héritage qui permet de créer des requêtes couplant plusieurs chemins est prise en compte à travers la notion de bloc de requêtes. La requête globale créée doit alors renvoyer l'union des résultats renvoyés par chaque sous requête. Dans notre cas, nous appelons parallélisation, le passage par une relation d'héritage divisant un chemin en 2 chemins ou plus.

La recherche de relations d'héritage se fait de manière récursive. Cela signifie qu'au sein d'une relation d'héritage, il est possible de détecter et de prendre en compte d'autres relations d'héritage. Cette détection récursive implique qu'il peut y avoir une parallélisation de la requête à différents niveaux. Il faut donc être capable de différencier les chemins issus de la même relation d'héritage et les chemins issus d'autres relations d'héritage.

La notion de bloc est une notion que nous avons créée pour répondre à ce problème. Un bloc correspond à un chemin issu d'une relation d'héritage et allant d'un nœud correspondant à la relation spécialisée, jusqu'au nœud correspondant à la relation annotée avec le concept de sortie.

Toutes les informations pour gérer les blocs sont déjà présentes dans la première version de la requête créée dans la partie précédente. En effet, en plus des lignes contenant un triplet, des lignes contenant des informations sur les blocs ont été ajoutées. Il y a 2 types de relation entre blocs à prendre en compte:

- des blocs de même niveau: 2 blocs sont considérés de même niveau s'ils sont issus de la même relation d'héritage. Dans les triplets créés précédemment une telle relation entre blocs est représentée par une ligne dont la valeur est "UNION". Cette relation entre bloc doit conduire, dans la requête finale, au renvoi de l'union des données renvoyées par chacun des blocs de même niveau.

- des blocs de niveau différent: le changement de niveau de blocs implique le passage par une relation d'héritage. Dans les triplets créés précédemment une telle relation entre blocs est représentée par une ligne prenant la valeur "fin de bloc". Cette relation entre blocs doit conduire, dans la requête finale, à la création d'un groupe de sous blocs; le parcours d'un bloc de ce type se traduit donc dans la requête finale par la présence de crochets ouvrant "{" ou de crochets fermant "}".

Pour réaliser la requête finale, un objet Java nommé "Bloc" a été créé. Un objet Bloc contient les informations suivantes:

- **le niveau du bloc**: Ce niveau correspond au nombre de relations d'héritages parcourues. Par exemple, une requête n'ayant aucune relation d'héritage n'aura qu'un bloc de niveau 0. Si une relation d'héritage est parcourue, les blocs issus de cette relation seront de niveau 1. Si une 2ème relation d'héritage est parcourue pour un bloc étant déjà de niveau (c'est à dire un bloc déjà issu d'une relation d'héritage), le bloc résultant sera de niveau 2.

- **les types de bloc**: Un bloc peut être de trois types. Soit il s'agit du premier bloc de la requête, soit il s'agit d'un bloc de niveau différent que celui du bloc précédent, soit il s'agit d'un bloc du même niveau que le bloc précédent.

- **les triplets correspondant au bloc**: tableau contenant tous les triplets du bloc. C'est ces triplets qui vont devoir être intégrés dans la requête finale en prenant en compte le niveau et le type de bloc.

L'utilisation des objets Bloc va permettre de chorégraphier l'insertion des blocs dans la requête.

Dans le cas de notre exemple de la Figure 66, 3 blocs seront créés. Le premier bloc comporte les triplets allant du concept d'entrée jusqu'à la détection d'une relation de spécialisation. Aucune relation d'héritage n'a été parcourue avant ce bloc, il s'agit donc d'un

bloc de niveau 0. Les 2 autres blocs sont issus de la relation d'héritage et correspondent aux 2 chemins résultant de cette relation. Les triplets de chacun de ces blocs contiennent le chemin allant de la relation de spécialisation jusqu'au concept de sortie.

La chorégraphie des blocs pour la création de la requête issue de notre exemple est présentée dans les Tableau 3 et Tableau 4:

	état initial	étape 1	étape 2
bloc 1	0	0	0
bloc 2	1	1	
bloc 3	1		

Tableau 3 : Evolutions du niveau des blocs au cours des étapes de la chorégraphie

	étape initiale	étape 1	étape 2

bloc 1	triplets 1	triplets 1	triplets1
			{
			triplets2
			UNION
			triplets3
			}

bloc 2	triplets 2	triplets2 UNION triplets3	
bloc 3	triplets 3		

Tableau 4 : Evolution des triplets contenus dans chaque bloc au cours des étapes de la chorégraphie

La chorégraphie des blocs pour notre exemple est réalisée en 2 étapes. Initialement, chaque bloc possède son niveau et ses triplets.

Lors de la première étape, les blocs ayant le plus haut niveau (bloc 2 et bloc 3) fusionnent. La fusion est reportée dans le bloc 2. Le niveau de ce bloc reste le même et les triplets des 2 blocs sont séparés par un UNION.

Lors de la 2ème étape, les blocs numéro 1 et 2 n'ont pas le même niveau. On réalise donc une fusion de ces blocs mais cette fois ci, les triplets du bloc 2 ne sont pas ajoutés sous forme d'union mais sous forme de sous partie de requête. Dans ce cas le niveau du bloc résultant récupère le niveau le plus petit des 2 blocs, ici 0. Il ne reste plus qu'un bloc de niveau 0, la chorégraphie est donc terminée.

Le pseudo code de l'algorithme utilisé pour la chorégraphie des blocs est représenté ci dessous:

```
Entrée
vecteurBloc : vecteur de type Bloc, comportant tous les blocs de la requête
Sortie
vecteurBloc : vecteur de type Bloc comportant tous les blocs correctement chorégraphiés
Paramètres
niveauMax : niveau le plus haut atteint par un bloc du vecteur
pos : position dans le vecteur du bloc ayant le plus haut niveau
booleenPremierPredicat : booléen
booleenDernierPredicat : booléen
Fonction detectionDeBlocs
tant que (vecteurBloc possède plus d'un élément)
 levelMax = niveau le plus haut d'un bloc
 position = position du bloc de plus haut niveau
 blocCourant = bloc du vecteur vecteurBloc à la position pos
 blocSuivant = bloc du vecteur vecteurBloc à la position pos+1
 si (blocCourant et blocSuivant sont de même niveau)
  triplet = nouveau vecteur de String
  triplet.ajoute(blocCourant.triplets)
  triplet.ajoute("}")
  triplet.ajoute("UNION")
  triplet.ajoute("{")
  triplet.ajoute(blocSuivant.triplets)
  nouveauBloc = nouveau objet de type Bloc
  nouveauBloc.niveau = niveauMax
  nouveauBloc.triplets = triplet
  nouveau
  supprime élément de vecteurBloc aux positions pos et pos+1
  ajoute l'élément nouveauBloc à vecteurBloc à la position pos
 fin si
 sinon
  si (blocCourant.triplets contient "UNION")
   positionUnion = position du dernier "UNION" des triplets du bloc courant
   premierPredicat = prédicat du triplet à la position positionUnion+2
   dernierPredicat = dernier prédicat des triplets du blocCourant
   triplet = nouveau vecteur de String
   booleenPremierPredicat = false
   booleenDernierPredicat = false
   pour (i allant de 0 à la taille de blocSuivant.triplets)
    si(booleenPremierPredicat == false)
     predicatCourant = prédicat de blocSuivant.triplets[i]
     si (predicatCourant == premierPredicat)
      booleenPremierPredicat = true
      triplet.ajoute("{")
      triplet.ajoute(blocCourant.triplets)
      triplet.ajoute("}")
     fin si
     sinon
      triplet.ajoute(blocCourant.triplets[i])
     fin sinon
    fin si
    sinon si (booleenDernierPredicat == false)
     predicatCourant = prédicat de blocSuivant.triplets[i]
     si (predicatCourant == dernierPredicat)
      booleenDernierPredicat = true
```

```
        fin si
      fin sinon si
      sinon
        triplet.ajoute(blocCourant.triplets[i])
      fin sinon
    fin pour
    nouveauBloc = nouveau objet de type Bloc
    nouveauBloc.niveau = niveauMax – 1
    nouveauBloc.triplets = triplet
    supprime élément de vecteurBloc aux positions pos et pos+1
    ajoute l'élément nouveauBloc à vecteurBloc à la position pos
  fin si
  sinon
    supprime élément de vecteurBloc à la position pos
  fin sinon
 fin sinon
fin tant que
fin Fonction detectionDeBlocs
```

Suite à cette étape, la structure de la requête résultante est correcte. Cependant, elle n'est pas encore compatible avec le formalisme SPARQL.

8.4. Création de la requête SPARQL résultant de l'algorithme

La dernière étape de notre création de requête consiste à rendre notre requête compatible avec le formalisme SPARQL. Pour cela, certaines modifications sont nécessaires.

Ajout et gestion des espaces de nom :

En SPARQL, les espaces de noms doivent être déclarés avant le début de la requête sous la forme :PREFIX gcpdm: <http://gcpdomainmodel.org/GCPDM#>
où gcpdm correspond au préfixe utilisé dans la requête pour l'espace de nom http://gcpdomainmodel.org/GCPDM#. Dans le corps de la requête, l'espace de nom ne doit pas apparaitre mais être représenté par le préfixe correspondant. Nous avons donc récupéré l'ensemble des espaces de noms utilisés dans la requête puis nous avons ajouté les déclarations correspondantes dans l'entête de la requête. Nous avons ensuite supprimé tous les espaces de nom présents dans le corps de la requête et nous les avons remplacés par les préfixes correspondants.

Première ligne du corps de la requête

Tout comme les requêtes SQL, les requêtes SPARQL possèdent une ligne permettant de déterminer le type d'action à réaliser ainsi que les paramètres que la requête doit renvoyer. SPARQL permet de créer des requêtes constructives en utilisant le mot clé CONSTRUCT ou

des requêtes interrogatives en utilisant le mot clé SELECT. Nos requêtes sont de type interrogatif et renvoient les paramètres correspondants aux concepts ontologiques d'entrée et de sortie. Nous rajoutons également le mot clé DISTINCT qui permet d'éviter les doublons dans les résultats renvoyés. La ligne rajoutée sera donc de la forme :

```
SELECT DISTINCT ?entree ?sortie WHERE {
```

Balise FILTER

Dans nos requêtes nous souhaitons filtrer les données de sortie en fonction de la donnée prise en entrée. Dans le langage SPARQL, la balise FILTER permet de réaliser un filtre sur un paramètre de la requête. Cette balise peut être utilisée pour filtrer sur la valeur d'un paramètre, par exemple sélectionner tous les paramètres inférieurs à une certaine valeur. Elle peut également être utilisée pour filtrer sur une expression régulière. Dans notre implémentation de requête nous avons décidé d'utiliser un filtre avec une expression régulière. Cela nous permet par exemple de renvoyer tous les marqueurs utilisés dans une étude dont le nom commence par la lettre A (avec l'expression régulière « ^A.* ». Dans nos requêtes un seul filtre sera présent. Il concernera le paramètre d'entrée de la requête. La ligne ajoutée sera donc de la forme :

```
FILTER regex(?entree,"^expression_reguliere$").
```

La requête résultant de notre exemple est présentée dans la Figure 72. Dans cette requête on peut clairement voir la présence des deux chemins parcourus. Les résultats renvoyés par cette requête seront l'union des résultats renvoyés par chacun de ces chemins.

```
SELECT DISTINCT ?study_name ?germplasm_name WHERE {
 ?study_id gcpdm:study ?study_name.
 FILTER regex(?study_name,"^name_of_the_study$").
 {
    ?genotypestudy_id vocab:genotypingstudy_id_study ?study_id.
    ?key vocab:dnasamplegenotypingstudy_id_genotypingstudy ?genotypestudy_id.
    ?key vocab:dnasamplegenotypingstudy_id_dnasample ?dnasample.
    ?dnasample vocab:dnasample_id_germplasm ?germplasm_id.          chemin 1
    ?germplasm_id gcpdm:germplasm ?germplasm_name.
 }
 UNION {
    ?phenotypestudy_id vocab:phenotypingstudy_id_study ?study_id.
    ?key vocab:germplasmphenotypingstudy_id_phenotypingstudy ?phenotypestudy_id.
    ?key vocab:germplasmphenotypingstudy_id_germplasm ?germplasm_id.   Chemin 2
    ?germplasm_id gcpdm:germplasm ?germplasm_name.
  }
}
```

Figure 72 : Requête SPARQL finale

8.5. Résultats

Nous allons, dans un premier temps, vérifier que notre approche permet de créer des requêtes renvoyant plus de résultats qu'une requête créée en utilisant un algorithme de Dijkstra. Nous vérifierons ensuite la cohérence des requêtes créées en comparaison à des requêtes SQL créées manuellement. Nous finirons en présentant des benchmarks de temps de création et d'exécution des requêtes, ainsi que le temps d'exécution de SWS contenant une requête créée automatiquement.

8.5.1. Pertinence des requêtes traitées par l'algorithme

La prise en compte des relations d'héritage, des tables d'association et des arités a pour but d'augmenter la cohérence des requêtes créées automatiquement. Une requête cohérente est une requête renvoyant exactement les mêmes résultats qu'une requête créée manuellement. Pour tester la cohérence des requêtes que nous avons créées nous allons, dans un premier temps, comparer les requêtes créées à l'aide de notre algorithme, aux requêtes qui auraient été créées en utilisant l'algorithme de Dijkstra.

Les requêtes présentées ici sont des requêtes posées sur la base de données relationnelle TropGene (Ruiz et al. 2004) et plus précisément sur le module contenant les données sur le riz. Notre choix c'est porté sur cette base de données relationnelle car elle a été développée au CIRAD, est accessible sur les serveurs du CIRAD, et qu'il est possible d'en faire facilement une copie locale. . Elle présente surtout l'avantage d'être une des sources de données utilisée par l'application web d'analyse de diversité génétique pour des populations de plantes GenDiversity. Dans le cadre de cette application, des Services Web ont déjà été créés pour interroger TropGene. Il nous sera donc possible de comparer le temps d'interrogation de Services Web créés manuellement à celui de nos Services Web Sémantiques créés automatiquement. Les requêtes utilisées ainsi que le schéma de la base de données relationnelle Tropgene sont présents dans l'annexe C. Le Tableau 5 ci dessous présente les caractéristiques des 5 requêtes utilisées pour nos benchmark. La colonne héritage indique si le plus court chemin traverse un nœud correspondant à une spécialisation. La colonne *même nombre de nœuds* indique si plusieurs plus courts chemins, reliant l'entrée à la sortie, présentent le même nombre de nœuds intermédiaires. Ce paramètre va permettre de

vérifier que dans ce cas précis, la prise en compte de la spécificité des relations du modèle relationnel va permettre à notre algorithme de créer des requêtes plus cohérentes.

	héritage	même nombre de nœuds
requête 1	oui	non
requête 2	non	oui
requête 3	non	oui
requête 4	non	non
requête 5	oui	non

Tableau 5 : Caractéristiques des requêtes

Comme détaillé précédemment, nous nous basons sur l'hypothèse que l'utilisation de plus court chemin dans un graphe où les nœuds sont reliés par des clés étrangères évite de renvoyer des résultats non souhaités. L'utilisation d'un seul chemin empêche également la redondance des données renvoyées. Le but de notre approche est uniquement d'augmenter le nombre de résultats renvoyés.

8.5.1.1.Intérêts de notre approche

Dans un premier temps, nous allons utiliser ces requêtes pour déterminer si les conditions ajoutées dans notre algorithme de recherche de plus court chemin permettent d'augmenter le nombre de résultats renvoyés par rapport à des requêtes créées en utilisant l'algorithme de Dijkstra. Comme nous avons posé l'hypothèse que l'ensemble des données renvoyées par une requête créée automatiquement sont pertinents, le renvoi d'un plus grand nombre de données par une requête, implique que cette requête est plus pertinente. Le Tableau 6 ci dessous montre le nombre de données renvoyées. La colonne nommée Dijkstra contient le nombre de données renvoyées par une requête créée automatiquement en utilisant l'algorithme de Dijkstra. La colonne nommée *avec conditions* contient le nombre de données renvoyées par une requête créée automatiquement en utilisant notre algorithme pour détecter le plus court chemin, ou la combinaison de plus courts chemins.

	Dijkstra	avec conditions	gain
requête 1	1595	7212	4,5
requête 2	0	12302	-
requête 3	197	197	1,0
requête 4	2055	2055	1,0
requête 5	1595	1595	1,0

Tableau 6 : Comparaison du nombre de données renvoyées par les requêtes créées en utilisant l'algorithme de Dijkstra ou notre algorithme

Les requêtes 1 et 2 créées en suivant notre approche renvoient plus de résultats que celles créées en suivant un algorithme de Dijkstra. La requête 4 ne possède ni relation d'héritage ni chemins ayant le même nombre de nœuds. Il s'agit d'une requête témoin car dans ce contexte les 2 algorithmes sont sensé trouver le même plus court chemin. Comme nous l'attendions, cette requête renvoi le même nombre de résultat pour les 2 approches. Les requêtes 3 et 5 contiennent respectivement 2 chemins ayant le même nombre de nœuds et une relation d'héritage. Malgré cela, notre approche ne renvoie pas plus de résultats qu'une requête créée en suivant un algorithme de Dijkstra. Le même nombre de résultats renvoyés dans le cas de la requête 3 peut indiquer que dans ce cas ou 2 plus courts chemins identiques étaient possible, l'algorithme de Dijkstra a involontairement sélectionné le chemin le plus cohérent. Dans le cas de la requête 5, notre algorithme aurait du trouver plus de résultats que l'algorithme de Dijkstra. Cela est d'autant plus bizarre que la seule différence entre les requêtes 1 et 5 est l'inversement des concepts d'entrée et sortie.

8.5.1.2.Comparaison avec les requêtes créées manuellement

Nous allons maintenant comparer le nombre de données renvoyées par une requête SPARQL créée avec notre algorithme, par rapport à une requête SQL créée manuellement par un expert. Les requêtes créées manuellement sont les requêtes les plus pertinentes possibles. Dans ces conditions, nous testons la pertinence des requêtes créées automatiquement. Si le nombre de données renvoyées par les requêtes créées manuellement est identique au nombre de données renvoyées par les requêtes créées automatiquement par notre approche, nos requêtes sont alors considérées comme pertinentes. Cela nous permettra de voir si notre approche peut être utilisée pour créer automatiquement des requêtes, ou bien si le nombre de données manquantes est trop important pour utiliser notre approche. Dans le Tableau 7 ci dessous, la colonne données renvoyées correspond au nombre de données renvoyées par une requête créée automatiquement en utilisant notre algorithme de recherche de plus court chemin. La colonne données attendues contient le nombre de données renvoyées par une requête créée manuellement et la colonne gain contient le ratio de données attendues par rapport aux données renvoyées.

	données renvoyées	données attendues	gain
requête 1	7212	7212	1,0
requête 2	12302	12302	1,0
requête 3	197	197	1,0
requête 4	2055	2055	1,0
requête 5	1595	7212	4,5

Tableau 7 : Comparaison entre les données renvoyées par une requête créée manuellement et une requête créée automatiquement

Les requêtes 1 à 4 renvoient le nombre attendu de résultats. Notre approche permet donc de créer automatiquement les requêtes attendues. Par contre la requête 5 renvoie 4,5 fois moins de résultats qu'attendu. Notre approche ne permet donc pas actuellement d'assurer que la requête créée est pertinente. La non pertinence de la requête 5 est du au fait que l'algorithme ne supporte pas actuellement la détection de relations d'héritage lors du parcours d'un graphe dans le sens de la table généraliste vers les tables spécialisées. Ce défaut pourra être corrigé en rajoutant une condition supplémentaire à la recherche de plus court chemin. Il va donc falloir retravailler notre implémentation pour tenir compte de ce cas auquel nous n'avions pas pensé initialement.

En dehors du cas des relations d'héritages parcourues dans le sens relation spécialisée vers relation généraliste, toutes les requêtes créées renvoient 100% des données attendues.

8.5.2. Benchmark sur les temps de création et d'exécution des requêtes

Les requêtes SPARQL sont créées automatiquement, tendent à être pertinentes, et sont ajoutées automatiquement aux SWS créés dans la plateforme d'intégration BioSemantic.

Dans cette partie nous allons présenter des tests effectués sur les temps nécessaires à la création et à l'exécution des requêtes.

8.5.2.1.Estimation du temps de création de la requête

La création de requête comporte plusieurs étapes.

- sélection des vues RDF annotés avec les concepts d'entrée et de sortie

- recherche d'un chemin pour chaque fichier sélectionné
- transformation du chemin en requête

Le temps nécessaire à la création de requête dépend fortement du type de chemin détecté ainsi que de la taille de la vue RDF du schéma de la base de données. Plus la base comporte de tables, plus la recherche de plus court chemin sera consommatrice de temps. Le Tableau 8 rapporte le temps nécessaire à la création de requêtes sur 2 bases de données relationnelles différentes:

- Tropgene, une base de données relationnelle contenant 90 tables et 15 millions d'enregistrements
- OrygenesBD, une base de données relationnelle contenant 11 tables et 22 millions d'enregistrements

Bien que la requête SPARQL ne soit créée qu'une seule fois, durant l'étape de génération de SWS, il est intéressant de mesurer le temps nécessaire à sa création

base de données	**relation d'héritage**	**longueur du chemin**	**temps en sec, (± 0.1)**
OrygenesDB	non	2 nœuds	**1.2**
Tropgene	non	4 nœuds	**2.0**
Tropgene	oui	4-3-2-6 nœuds	**14.6**

Tableau 8 : Estimation du temps nécessaire à la création de requête SPARQL.

Le Tableau 8 montre des temps nécessaires à la création automatique d'une requête SPARQL. Les 2 premières lignes du tableau permettent de voir que, malgré la grande différence de nombre de tables entre les 2 bases de données relationnelles, le temps nécessaire pour la création de requêtes sur les 2 bases est très proche, et assez court. Par contre, la création d'une requête comportant une relation d'héritage, est beaucoup plus longue. Dans la dernière ligne de notre tableau, la requête créée comporte une relation d'héritage et a nécessité la création d'une requête composé de 4 chemins. Ces chemins contiennent respectivement 4, 3, 2 et 6 nœuds intermédiaires. La prise en compte d'une relation d'héritage est très consommatrice de temps. Dans notre exemple, 14.6 secondes sont nécessaires à la création d'une seule requête. Le temps de création d'une requête peut donc être plus ou moins long mais reste de l'ordre de la seconde. Cette étape n'est donc pas un frein à la création de SWS.

8.5.2.2.Estimation du temps d'exécution de la requête

Les sources de données biologiques contiennent une grande quantité de données. Bien que les requêtes renvoient les données souhaitées, la quantité de données à renvoyer pourrait induire un temps d'exécution très important, pouvant nuire à l'intérêt de la plateforme. L'augmentation du temps d'exécution peut avoir 2 causes principales. La première est l'utilisation de la plateforme D2RQ pour interroger les bases de données relationnelles. Les requêtes SPARQL sont ainsi transformées en requêtes SQL puis les résultats sont renvoyés sous forme de triplets. La transformation peut être consommatrice de temps ou induire la création de requêtes SQL non optimisées. Nous avons donc décidé de mesurer la perte de performance d'exécution de nos requêtes SPARQL. Cette perte de performance est déterminée en comparant le temps nécessaire à l'exécution d'une requête SQL créée manuellement, par rapport au temps d'exécution d'une requête SPARQL correspondante, créée automatiquement. Le Tableau 9 compare le temps d'exécution de 5 requêtes de la base de données Tropgene. Le temps nécessaire à l'exécution des requêtes SQL est mesuré avec éclipse en utilisant la librairie java.sql. Le temps nécessaire à l'exécution des requêtes SPARQL est quand à lui mesuré sous l'explorateur SPARQL de D2R Server.

L'exécution de requêtes SPARQL met en moyenne 3 à 4 fois plus de temps que l'exécution de requêtes SQL. Cependant, il est tout de même possible de retourner plus de 5000 enregistrements en quelques secondes. La grande différence entre les temps d'exécution peut s'expliquer de 2 manières. La première est le fait que les données renvoyées dans D2R Server sont également affichées. Le temps d'exécution enregistré dans notre benchmark comprend donc le temps d'exécution ainsi que le temps d'affichage des données renvoyées, ce qui n'est pas le cas pour les requêtes SQL. En effet, pour les requêtes SQL, seul le temps d'exécution est pris en compte.

Nombre de tables	**relation d'héritage**	**longueur du chemin**	**Nombre de résultats**	**Requête SQL (secondes, ± 0.1)**	**Requête SPARQL dans D2R Server (secondes, ± 0.1)**
90	non	4	860	**0.3**	**1.4**
90	non	4	1456	**0.4**	**1.4**
90	non	3	2055	**0.8**	**2.3**
90	oui	4-3-2-6	8071	**1.1**	**4.2**
90	non	3	12302	**2.3**	**4.8**

Tableau 9 : Comparaison du temps d'exécution nécessaire entre une requête SQL créée manuellement et la requête SPARQL comparable créée automatiquement

8.5.2.3.Estimation du temps d'exécution du Service Web

La deuxième cause probable d'augmentation du temps d'exécution de nos requêtes SPARQL est l'architecture de la plateforme BioSemantic. Pour vérifier que la plateforme n'implique pas une chute des performances de requêtage, nous avons donc décidé de mesurer le temps d'exécution de différents Services Web. D'un coté nous allons mesurer le temps d'exécution d'un Service Web contenant une requête SQL, et de l'autre coté nous allons mesurer le temps d'exécution d'un SWS créé automatiquement à l'aide de la plateforme BioSemantic et contenant une requête SPARQL créée automatiquement.

La comparaison du temps nécessaire à l'exécution de Services Web créés manuellement par rapport à celui de nos SWS est présentée dans le Tableau 10. Ces Services Web interrogent la base de données Tropgene et sont utilisés sur le portail Web GenDiversity. Bien que les Services Web créés manuellement ont une exécution plus rapide, les SWS créés automatiquement dans par la plateforme BioSemantic ont un temps de réponse de l'ordre de la seconde. Le temps d'exécution n'affecte donc pas l'utilisation de nos SWS.

Requête	**nombre de résultats**	**Services Web SQL (secondes, ± 0.1)**	**SWS BioSemantic (secondes, ± 0.1)**
renvoie les études de génotypage	7	**0.2**	**1.0**
renvoie les germplasmes pour les études sélectionnées	860	**0.4**	**1.0**
renvoie les marqueurs pour les études sélectionnées	1456	**0.4**	**1.0**

Tableau 10 : Comparaison entre le temps nécessaire à l'exécution de services web SQL et les services web sémantiques créés par la plateforme BioSemantic.

8.6. Perspectives

La prochaine étape de notre travail consiste à prendre en compte un cas particulier n'étant pas traité par l'algorithme de Dijkstra, il s'agit du cas ou la table de sortie est la même que la table d'entrée. En effet, ce cas particulier est envisageable dans un graphe représentant un modèle relationnel, une table pouvant avoir une jointure sur elle-même. La recherche de ce type de chemin pourrait par exemple être intéressante dans le cas où on souhaite avoir un gène en entrée de notre requête et renvoyer en sortie tous les gènes reliés par une relation de type orthologue. Dans ce cas précis notre algorithme ne renvoi actuellement aucun chemin car, dans l'algorithme de Dijkstra, un nœud ayant été parcouru ne peut être parcouru une seconde

fois. Pour résoudre ce cas où les concepts d'entrée et sortie sont les mêmes, il serait possible d'implémenter un algorithme utilisé uniquement dans ce cas particulier et détectant seulement les jointures vers une même table.

La deuxième évolution très importante consiste à prendre en compte la possibilité d'avoir non pas une seule entrée mais une liste de données en entrée. Il s'agit d'un critère primordial à l'utilisation de nos requêtes dans la création de flux de travaux. Actuellement nos SWS ne peuvent prendre qu'un seul paramètre en entrée, ce paramètre pouvant être une expression régulière. La prise en compte de plusieurs paramètres d'entrée implique actuellement d'appeler le SWS à l'intérieur d'une boucle. Il serait intéressant d'intégrer cette fonctionnalité directement au sein de la requête SPARQL créée automatiquement. Cela peut être réalisé très facilement en rajoutant des conditions à notre FILTER. Par exemple, il serait possible sur plusieurs expressions régulières. Dans notre exemple la valeur associée au concept d'entrée peut finir par un 1 ou par un 2.

FILTER (regex(?study_name, "1$") || regex(?study_name, "2$")).

La plus grosse limitation de nos requêtes vient de la limitation à un concept d'entrée et un concept de sortie. Cette limitation est due au fait que les algorithmes de recherche de plus court chemins permettent uniquement de relier un nœud d'un graphe à un autre nœud ce qui implique que la requête que nous créons est issue d'un chemin linéaire dans le graphe représentant le schéma de la base de données. Plusieurs solutions d'amélioration sont envisageables pour pouvoir sélectionner plusieurs concepts d'entrée et plusieurs concepts de sortie. La première solution consisterait à détecter le plus court chemin pour chaque couple composé d'une entrée et d'une sortie. Il faudrait ensuite trouver en moyen cohérent de fusionner tous les chemins détectés en un graphe comportant le moins de nœuds possible. Le problème principal de cette approche vient du fait que lors de la détection de nos plus courts chemins, nous favorisons le passage par des tables d'associations binaires. Ce choix était justifié par la création de chemins linéaires. Dans ce type de chemin il n'était pas possible d'imaginer la fusion de plusieurs chemins en un nœud correspondant à une table d'association. Il n'est donc pas imaginable de réutiliser tel quel l'algorithme développé par notre approche. Il serait par contre envisageable de réutiliser une portion de notre algorithme mais en modifiant l'avantage lié au passage par une table d'association binaire. Notre algorithme pourrait être utilisé lors d'un premier passage de création de plus court chemin linéaire. Une deuxième étape consisterait à détecter les nœuds où plusieurs chemins linéaires fusionnent. Il serait ensuite possible de rajouter d'autres étapes tentant d'optimiser le plus court chemin global obtenu en cherchant par exemple à réaliser ces fusions dans des tables

d'associations n-aire, ou en tentant de diminuer le nombre de nœuds sur lequel sont réalisés des fusions. On pourrait également imaginer de compter le nombre n de fusions sur un nœud et favoriser ces fusions sur les tables d'association d'arité n+1 (n associations d'entrée et une association de sortie). Cependant, rien n'assure que le fait de se baser sur notre algorithme pour créer des requêtes comportant plusieurs entrées ou sorties permette la création de l'algorithme le moins complexe possible.

Dans notre approche l'annotation sémantique consiste à annoter des nœuds représentant des éléments du schéma de la base de données. Notre approche utilise les clés étrangères pour parcourir le graphe et passer d'un nœud représentant une table à un nœud représentant une autre table. Il serait intéressant d'ajouter la possibilité d'annoter des arcs correspondant à des clés étrangères. Cette évolution pourrait avoir plusieurs intérêts. Le premier d'entre eux serait la simple possibilité de visualiser les associations contenues dans la requête créée automatiquement. Cela éviterait de devoir regarder en détail la requête SPARQL pour vérifier qu'elle soit cohérente. On pourrait directement visualiser une sorte de logique de parcours de la requête. Le deuxième intérêt pourrait se situer dans les paramètres à rentrer lors de la création de requête. On pourrait en effet proposer à l'utilisateur de sélectionner une ou plusieurs annotations sémantiques parcourues lors de la création du plus court chemin. Enfin, la dernière possibilité de ce type d'annotations pourrait être la possibilité de modifier automatiquement les requêtes. Pour cela on peut imaginer un système permettant d'afficher les annotations de clés étrangères parcourues lors de la création d'une requête, avec la possibilité ; par exemple, de supprimer une annotation, forçant ainsi la détection d'une nouvelle requête ne passant pas par cette clé étrangère.

Notre algorithme de recherche de plus court chemin prend en compte les relations d'héritage entre 2 relations du schéma. Il pourrait être intéressant de rajouter d'autres règles lors de la création de la vue RDF. Certaines approches (Alalwan et al. 2009; Man Li et al. 2005) permettent de détecter semi-automatiquement des relations qui ont été partitionnées lors du passage du modèle conceptuel au modèle relationnel et permettent de les mapper à la même classe. L'utilisation de cette règle nécessiterait cependant une création semi-automatique de notre vue RDF avec des propositions de relations partitionnées à valider par un utilisateur. D'autres approches permettent la détection automatique d'entités faibles et leur annotation à l'aide des propriétés inverses *isPartOf* et *hasPart* (Astrova 2009). L'utilisation de telles règles pourraient être utilisées dans notre algorithme de recherche de plus court chemin en rajoutant des conditions de parcours supplémentaires.

8.7. Conclusion

Nous avons développé une méthodologie originale permettant de créer automatiquement des requêtes SPARQL. Cette méthode est basée sur le parcours de graphes et plus précisément sur la recherche de plus court chemin reliant deux nœuds d'un graphe. Le graphe utilisé est une vue RDF du schéma relationnel d'une base de données relationnelle. Le parcours de ce graphe se fait en utilisant les clés étrangères. Le plus court chemin correspondra donc au chemin nécessitant le parcours du plus petit nombre de clés étrangères permettant de relier 2 tables d'une base de données relationnelle. De nombreux algorithmes de recherche de plus court chemins existent mais aucun ne tient compte du contexte spécifique de la recherche de plus court chemin dans un schéma relationnel. Nous avons donc modifié l'algorithme de Dijkstra afin de créer un plus court chemin sous conditions prenant en compte ce contexte relationnel. Nous avons rajouté une étape d'ingénierie inversée permettant d'ajouter des informations sur les relations d'héritage lorsque celles-ci ne sont pas aplaties lors de la transformation du schéma entité association vers le schéma relationnel. Notre algorithme prend également en compte le passage par des tables d'association et l'arité de ces tables. Le plus court chemin sous condition détecté est ensuite transformé en requête SPARQL. Nous avons montré que les résultats renvoyés par ces requêtes créées automatiquement sont identiques aux résultats renvoyés par une requête SQL créée manuellement. Le temps d'interrogation de ces requêtes est plus important que celui d'une requête SQL mais il reste de l'ordre de la seconde. Il ne s'agit donc pas d'un facteur limitant dans leur utilisation.

Dans notre approche nous avons ajouté ces requêtes à des Services Web Sémantiques créés automatiquement par la plateforme BioSemantic. Cependant, le système étant générique, il est tout à fait possible de créer ces requêtes indépendamment de BioSemantic de manière à interroger des points d'entrées SPARQL via le Web des Données. Il est également envisageable, en modifiant l'implémentation de la requête, d'utiliser notre approche de plus court chemin modifié pour créer directement des requêtes SQL.

9. Synthèse et discussion

9.1. Synthèse

9.1.1. Utilisation de BioMoby à travers BioMoby Converter

L'objectif initial de ce travail de thèse était d'automatiser la création d'adaptateurs sémantiques de type Services Web Sémantiques (SWS) sous BioMoby pour la plateforme Pantheon. Nous avons décidé de nous focaliser sur la création d'adaptateurs permettant l'intégration de bases de données relationnelles. Pour atteindre cet objectif nous avons imaginé une plateforme de création d'adaptateurs comportant 3 étapes :

- Ajout d'une bioontologie de domaine dans BioMoby
- Mise en correspondance entre le schéma d'une base de données et une bioontologie
- Utilisation des 2 étapes précédentes pour créer un SWS BioMoby

Nous avons développé BioMoby Converter un outil présenté sous la forme d'un plugin pour le logiciel d'édition d'ontologies Protégé. L'utilisation conjointe de Protégé et BioMoby Converter permet de créer ou éditer une ontologie, visualiser une ontologie, ajouter une ontologie ou une portion d'ontologie dans l'ontologie de types de données BioMoby, supprimer des types de données de BioMoby, et exporter l'ontologie de types de données BioMoby dans une ontologie OWL. Les classes bioontologiques ajoutées dans BioMoby peuvent ainsi être utilisées pour annoter les SWS BioMoby. BioMoby Converter répond aux besoins de la première étape de l'approche imaginée pour automatiser la création d'adaptateurs.

Cependant, l'utilisation de BioMoby entraine une perte d'expressivité des ontologies. De plus, les SWS sont basés sur une structure de message spécifique à la plateforme BioMoby. Cela limite la généricité des adaptateurs en forçant les clients à être spécifiques de la plateforme BioMoby. Pour nous débarrasser de ces limitations nous avons donc décidé d'abandonner l'utilisation de BioMoby. Nous nous sommes orientés vers le développement d'une plateforme de création automatisée d'adaptateurs basés sur les standards du Web Sémantique et sur les technologies des Services Web.

9.1.2. Vers une plateforme de création automatisée de Services Web Sémantiques

Nos objectifs n'ont pas changé avec l'abandon de BioMoby. Nous avons donc développé la plateforme BioSemantic automatisant au maximum la création d'adaptateurs de type Services Web Sémantiques.

Dans notre approche, une vue RDF du schéma relationnel d'une base de donnée est créée automatiquement en utilisant D2RQ. Cette vue est considérée comme une ontologie locale. Elle est annotée sémantiquement à l'aide de concepts ontologiques de manière à la mettre en correspondance avec une ontologie globale, représentée par une bioontologie. Ces annotations sont ensuite utilisées pour créer automatiquement un Service Web annoté sémantiquement par des balises SAWSDL. Enfin, ces SWS sont enregistrés dans BioCatalogue. Les annotations sémantiques et l'enregistrement dans BioCatalogue permet de détecter facilement un SWS. Elles permettent également la création de flux de travaux en combinant des SWS ayant le même concept ontologique d'entrée ou ayant un concept de sortie identique au concept d'entrée d'un autre SWS.

L'application Web de la plateforme BioSemantic a été développée pour offrir une interface graphique conviviale permettant de créer une vue RDF, charger une vue RDF dans un annuaire et créer des SWS.

Comme cas d'utilisation biologique, nous avons utilisé ces SWS pour intégrer des bases de données relationnelles à l'application GenDiversity.

9.1.3. Création automatisée de requêtes SPARQL

La dernière partie de ce travail de thèse a consisté à créer automatiquement des requêtes SPARQL. Une requête peut ainsi être intégrée dans un SWS, rendant la création de ce dernier totalement automatique. Notre approche originale de création de requêtes est basée sur une approche de recherche de plus court chemin dans un graphe, basée sur l'algorithme de Dijkstra, en prenant en compte les particularités d'un schéma relationnel. Nous avons utilisé une technique d'ingénierie inverse permettant de détecter les relations d'héritages initialement présentes dans le schéma entité association. Nous avons également détecté les tables d'association ainsi que l'arité de ces tables. Les informations détectées sont utilisées pour créer le plus court chemin. Ce plus court chemin est ensuite parcouru pour créer la requête SPARQL correspondante.

Nous avons montré que notre approche permet de créer des requêtes renvoyant les données identiques à une requête créée manuellement. Nous avons également montré que le temps d'interrogation de ces requêtes n'est pas un facteur limitant à leur utilisation.

9.2. Discussion

9.2.1. BioMoby Converter

Le plugin BioMoby Converter permet d'enregistrer automatiquement une ontologie OWL dans une ontologie de types de données BioMoby. Pour cela les classes de l'ontologie OWL sont préalablement triées dans un ordre compatible avec leur enregistrement dans BioMoby. L'étape de tri n'est pas consommatrice de temps en comparaison du temps nécessaire à l'enregistrement vers BioMoby. En effet, la communication avec un annuaire BioMoby lors d'enregistrement de types de données, représente la majorité du temps nécessaire pour l'enregistrement. Ce temps d'accès est dépendant de l'annuaire BioMoby, il ne peut donc pas être amélioré.

Il aurait été intéressant de coupler BioMoby Converter avec un outil de mise en correspondance entre ontologie et bases de données relationnelle. Datamaster et Maponto sont 2 plugins pour Protégé permettant de créer automatiquement une ontologie locale OWL à partir du schéma relationnel d'une base de données. L'utilisation d'un de ces plugins aurait permis de pouvoir créer automatiquement une ontologie locale. Cette ontologie aurait ainsi pu être utilisée dans un second plugin pour Protégé permettant de créer des correspondances avec une bioontologie. Nous aurions ainsi regroupé l'ensemble des étapes de mise en correspondance (base de données et BioMoby) dans le logiciel Protégé. L'abandon de l'utilisation de BioMoby nous a cependant conduits à utiliser D2RQ plutôt que ces plugins. Cela s'explique par les fonctionnalités supplémentaires proposées par D2RQ. En effet, D2RQ permet d'utiliser l'ontologie locale, créée automatiquement, pour créer un point d'entrée SPARQL. D2RQ intègre également un algorithme permettant de transformer une requête SPARQL posée sur l'ontologie locale (vue RDF) en une requête SQL interrogeant directement la base de données relationnelle.

BioMoby Converter prend en entrée des ontologies OWL. Il serait intéressant d'ajouter la possibilité de prendre en entrée des ontologies RDF ou des schémas XML. La prise en compte de Schéma XML permettrait notamment d'intégrer automatiquement le schéma XML créé dans la cadre du projet BioXSD. Ce schéma se veut un format d'échange de types de données standard en biologie moléculaire. Il possède une composante sémantique

car il est annoté à l'aide de balises SAWSDL. L'ajout du schéma BioXSD permettrait donc à BioMoby d'être compatible avec ce format d'échange standard.

9.2.2. BioSemantic

9.2.2.1.Comparaison avec d'autres approches de création de SWS

Nous comparerons BioSemantic avec les approches BioMoby (M. Wilkinson et al. 2005), SADI (M. D. Wilkinson et al. 2009) et SSWAP (D. D. G. Gessler, G. S. Schiltz, et al. 2009) qui ont en commun la création de Services Web Sémantiques biologiques. Dans cette partie seule les caractéristiques de création et d'utilisation des SWS des différentes approches seront comparées car, en dehors de BioSemantic, aucune de ces approches ne se place dans le contexte de la création automatisée d'adaptateurs spécifiques aux bases de données relationnelles.

Nous allons comparer ces approches selon 7 axes : i) l'utilisation exclusive de standards du Web Sémantique, ii) le type d'annotation des entrées et sorties des SWS, iii) le type d'architecture utilisée par les SWS iv) le besoin des clients d'être spécifiques à une plateforme v) la capacité de la plateforme à raisonner, vi) le degré d'automatisation de la création et du déploiement des SWS et iiv) le degré d'automatisation de la création de requête. Cette comparaison est présentée dans le Tableau 11.

Approches	Standards Web Sémantique	type d'annotations	architecture	plateforme spécifique	raisonneur	création/ déploiement	création des requêtes
BioMoby	non	XML	WSDL	oui	non	semi-automatique	manuel
SSWAP	oui	OWL	SSWAP	oui	oui	manuel	manuel
SADI	oui	OWL	SADI	non	oui	semi-automatique	manuel
BioSemantic	oui	SAWSDL	WSDL	non	non	automatique	automatique

Tableau 11 : Comparatifs des approches de création de Services Web Sémantiques

BioMoby est la seule approche, parmi celles comparées, à ne pas être basée sur des **standards du Web Sémantique**. La sémantique de cette approche provient des types de données stockés dans une arborescence XML. L'utilisation d'un même type de données pour deux Services Web différents facilite leur composition. Les approches SSWAP et SADI sont basés sur des annotations OWL. BioSemantic utilise quand à lui des annotations SAWSDL ajoutant ainsi une annotation sémantique à la description syntaxique initiallement présente dans les fichiers WSDL.

L'utilisation de classes OWL par SADI et SSWAP est rendue possible par la création de SWS non basée sur les technologies standards des Services Web. En effet, l'approche SSWAP a développé une nouvelle **architecture**, de nouveaux protocoles et une plateforme afin de rendre ses SWS plus léger que des Services Web développés sur les standards SOAP/WSDL. Dans l'approche SADI, une nouvelle structure de SWS a été développée afin de se baser uniquement sur RDF et OWL, apportant ainsi une richesse sémantique à tous les niveaux du SW. BioMoby et BioSemantic sont quand à eux basés sur la création de SWS utilisant les protocoles standards SOAP/WSDL.

Certaines de ces approches sont **plateforme spécifiques**. Dans l'approche SSWAP, cette spécificité est liée à l'utilisation de nouveaux protocoles devant être utilisés lors du développement de clients. Il s'agit d'une limite à SSWAP car les avantages obtenues en termes d'expressivité et de légèreté d'utilisation sont au prix d'une perte de généricité. Dans BioMoby des types de données spécifiques à la plateforme sont utilisés pour structurer les données renvoyées par chaque Service. De ce fait, les clients doivent être spécifiques à BioMoby afin d'être compatibles avec la structure des données renvoyées. Les approches SADI et BioSemantic ne nécessitent pas le développement de clients plateforme spécifiques.

SADI et SSWAP sont basés sur des annotations OWL et incluent des **raisonneurs logiques**. Cela permet par exemple, dans le cas de la recherche d'un SWS dont l'entrée est annotée avec une classe OWL précise, de détecter tous les SWS annotés avec une classe correspondant à une sous classe de celle souhaitée. La faible expressivité des types de données BioMoby ne permet pas se genre de raisonnement. BioSemantic dans son état actuel n'intègre pas de raisonneur. Les annotations SAWSDL pourraient être utilisées pour raisonner mais nous avions initiallement prévu de laisser cette tâche à l'annuaire dans lequel les SWS seraient référencés. Cette option n'est malheureusement pas disponible dans l'annuaire BioCatalogue.

Les deux derniers axes de comparaison définissent le degré d'automatisation de ces approches.

La **création et le déploiement** des SWS BioMoby sont réalisés de manière semi-automatique. L'interface graphique Dashboard permet de sélectionner différents paramètres du Service et de les utiliser pour en créer automatiquement un squelette. La création d'un SWS SADI est basée sur le même type d'approche semi-automatique. BioSemantic est la seule approche automatisant totallement cette étape.

Le développement de la **requête** du SWS est réalisé manuellement dans toutes les approches sauf BioSemantic.

9.2.2.2.Les étapes manuelles à améliorer

Actuellement notre plateforme d'intégration comporte deux parties manuelles : l'annotation des vues RDF et l'enregistrement des SWS dans BioCatalogue. Nous allons donc focaliser nos efforts sur ces deux parties de manière à les automatiser ou à les rendre plus accessibles.

Automatiser l'enregistrement des SWS

La prochaine étape de notre travail consistera à automatiser l'enregistrement de nos SWS dans BioCatalogue. Actuellement, l'enregistrement nécessite de passer par des interfaces graphiques pour rentrer le chemin vers le fichier WSDL ou encore prendre en compte les annotations sémantiques. Cela est une étape limitante dans notre plateforme car elle a été conçue pour créer et enregistrer automatiquement des SWS et ainsi favoriser leur découverte. Toutes les informations nécessaires à l'enregistrement automatique sont disponibles. BioCatalogue a publié une API permettant de manipuler les Services Web déjà existants dans leur annuaire. La prochaine version de cette API devrait intégrer la possibilité d'enregistrer et d'annoter des SWS (J. Bhagat, communication personnelle). Dès sa disponibilité, nos services pourront être enregistrés automatiquement.

Annoter automatiquement les vues RDF

Actuellement les annotations des vues RDF sont réalisées manuellement via un éditeur de texte. Cela signifie que l'annotateur doit i) être capable de comprendre un fichier RDF sérialisé en N3, ii) connaître le langage D2RQ, iii) avoir une connaissance du schéma de la base de données ainsi que des concepts ontologiques correspondants à chaque élément du schéma de la base de données. Cette annotation est une étape indispensable au bon fonctionnement de notre plateforme. Il faut donc absolument la rendre plus accessible aux fournisseurs de données. L'amélioration de l'étape d'annotation peut se faire de deux façons différentes. Dans un premier temps nous souhaitons créer une interface graphique

d'annotation conviviale permettant de créer des annotations en faisant simplement glisser un concept ontologique sur l'élément de la base de données que l'on souhaite faire correspondre. Cette approche aura l'avantage de permettre à des fournisseurs de données de réaliser une annotation du schéma de leur source sans avoir à ce soucier du format de la vue RDF ni du langage D2RQ utilisé pour créer les correspondances. La deuxième approche envisagée est une approche de détection automatique de correspondances. Cette détection automatique serait accompagnée de phases d'interactions avec l'utilisateur, lui permettant de valider ou refuser des correspondances. Pour cette approche automatique, nous sommes en collaboration avec l'équipe Zenith INRIA de Patrick Valduriez qui développe un logiciel de détection automatique de correspondances WebSmatch[17].

9.2.2.3.Améliorer l'expressivité des requêtes

La grande limitation de notre approche se situe au niveau de l'expressivité des requêtes que nous créons.

La plus grosse limite d'expressivité de nos requêtes vient de l'approche choisit pour les créer. L'utilisation du plus court chemin reliant deux nœuds d'un graphe nous permet uniquement de créer des requêtes prenant un attribut en entrée et renvoyant un attribut en sortie. Une perspective est d'élargir notre approche de manière à permettre la prise en compte de plusieurs colonnes en entrée comme en sortie.

9.2.2.4.Faciliter l'accès aux requêtes

Nos requêtes sont actuellement créées et intégrées immédiatement dans un SWS. Nous souhaitons rajouter la possibilité de visualiser cette requête lors de la phase de création du SWS. Cela permettra de vérifier sa véracité mais également de pouvoir la modifier avant de l'intégrer dans le SWS.

Ces requêtes ne sont accessibles que via l'utilisation des SWS. Il n'est donc pas possible actuellement de sélectionner des entrées et sorties pour créer une requête à utiliser sur un point d'entrée SPARQL. La détection puis la création de requêtes est une étape pouvant demander plus d'une dizaine de secondes pour une requête. Il ne serait donc pas envisageable de réaliser ces étapes pour chaque demande de création de requête. Il serait par contre envisageable de stocker les requêtes créées précédemment. Cela permettrait de visualiser et/ou utiliser rapidement une requête déjà créée.

[17] http://websmatch.gforge.inria.fr/

9.2.2.5.Optimiser la transformation SQL-SPARQL

Le temps d'exécution de nos SWS dépend du temps nécessaire à la récupération des données dans les sources d'origine, du temps nécessaire à la transformation de requêtes SPARQL en requêtes SQL mais également de l'optimisation de la requête SQL découlant de cette transformation. Le temps d'accès et de renvois des données provenant d'une source de données distante fait partie intégrante d'une approche de médiation. Ce temps d'accès est fixe et ne peut être amélioré par notre approche. Le temps nécessaire à la transformation d'une requête SPARQL vers une requête SQL est faible. Par contre, le temps d'interrogation des requêtes SQL créées automatiquement peut être très variable, selon leur optimisation. Les sources de données biologiques peuvent contenir une grande quantité de données et le temps d'exécution d'une simple requête SQL peut déjà être long. Il serait donc intéressant d'optimiser les requêtes SQL obtenues, de manière à diminuer leur temps d'interrogation. Plusieurs approches de transformation de requêtes SPARQL en SQL ont été publiées (Elliott et al. 2009; J. Lu et al. 2008; Bajda-Pawlikowski 2008; Chebotko et al. 2009). L'algorithme de transformation contenu dans D2RQ n'a pas été publié et ne mentionne aucune approche existante. Il serait donc intéressant de comparer le temps d'exécution de requêtes SPARQL transformées en SQL pour différentes approches. Cela permettrait de déterminer les performances de l'algorithme utilisé par D2RQ et le cas échéant d'utiliser une autre approche décrite dans la littérature.

Conclusion

Tout au long de ce travail de recherche nous avons eu pour objectif d'automatiser l'intégration de bases de données relationnelles en suivant une approche de médiation. Pour cela nous avons travaillé sur la création automatisée d'adaptateurs sémantiques de type Services Web Sémantiques.

Nos principales contributions

Dans un premier temps nous avons tenté d'automatiser l'approche mise en place pour la création d'adaptateurs sémantiques de la plateforme de médiation GCP Pantheon. La solution existante utilisait BioMoby pour créer des adaptateurs de type Service Web Sémantique. Nous avons mis en place une architecture permettant d'automatiser la création de ces Services Web Sémantiques (SWS) BioMoby en nous basant sur des bioontologies existantes. Cette architecture combinait trois étapes: l'enregistrement de bioontologies dans BioMoby, la mise en correspondance du schéma d'une base de données avec des bioontologies et la création automatisée de Services Web BioMoby. Nous avons développé **BioMoby Converter** un outil se présentant sous la forme d'un plugin pour le logiciel d'édition d'ontologie Protégé. BioMoby Converter permet d'enregistrer une ontologie OWL vers l'ontologie de types de données BioMoby, de supprimer des types de données enregistrés sous BioMoby, et d'exporter l'ontologie de types de données BioMoby vers une ontologie OWL.

La création de BioMoby Converter nous a fais prendre conscience des limitations techniques que nous imposait l'utilisation de BioMoby : son ontologie est basée sur une expressivité très faible, il n'utilise pas les standards du Web Sémantique et il est plateforme spécifique. Afin de nous libérer de ces limitations nous avons développé un système original de création automatisée de SWS.

Nous avons élaboré une plateforme possédant une architecture flexible permettant d'automatiser au maximum la création de SWS. Cette plateforme, nommée **BioSemantic**, est basée uniquement sur des standards du Web Sémantique. Dans cette plateforme une ontologie locale est créée automatiquement sous la forme d'une vue RDF du schéma d'une base de données relationnelle. La mise en correspondance entre les ontologies globales et les ontologies locale s'effectue par l'intermédiaire de l'annotation sémantique manuelle d'une vue RDF à l'aide de concepts issus de bioontologies. Ces vues RDF annotées sont utilisées

pour créer et déployer automatiquement des SWS dont les entrées et les sorties sont annotés sémantiquement à l'aide de balises SAWSDL. Ces SWS sont ensuite enregistrés dans l'annuaire se Services Web BioCatalogue et sont ainsi facilement identifiables par les utilisateurs. L'annotation des entrées et sorties permet également de faciliter la composition de ces Services Web à travers des flux de travaux.

Une application Web nommée **BioSemantic Application** (http://southgreen.cirad.fr/?q=content/biosemantic) a également été développée. Elle offre des interfaces graphiques simples permettant à un utilisateur de créer automatiquement une vue RDF puis d'utiliser toutes les vues RDF annotées sémantiquement pour créer automatiquement des SWS. Cette application offre une grande flexibilité d'utilisation car aucune connaissance du schéma d'une base de données ou des standards du Web Sémantique n'est nécessaire pour créer la vue RDF ou encore pour créer des SWS. Seule l'étape d'annotation de la vue RDF nécessite une action manuelle réalisée par un utilisateur expert ayant à la fois une connaissance précise du schéma de la base de données relationnelle, des bioontologies ainsi que du langage D2RQ utilisé pour créer la vue RDF.

Les SWS créés doivent permettre d'interroger des bases de données relationnelles. Pour que leur création soit automatique il a donc fallu développer une approche permettant de créer automatiquement des **requêtes SPARQL**. Cette création automatique est basée sur l'algorithme de Dijkstra de recherche de plus court chemins entre deux nœuds d'un graphe. Cet algorithme a été modifié pour tenir compte de notre contexte permettant la création automatique de requêtes interrogeant une base de données relationnelle. Les requêtes créées automatiquement renvoient les mêmes résultats que ceux obtenus par des requêtes créées manuellement. Toutefois, leur temps d'exécution est supérieur mais reste dans un ordre acceptable (inferieur a 15 secondes).

Perspectives

Bien que nous ayons décidé de ne pas utiliser BioMoby Converter en raison de trop nombreuses contraintes techniques, il serait intéressant d'ajouter des XML Schéma contenant des annotations sémantiques de type SAWSDL. Ces dernières sont recommandées par le W3C car elles permettent la mise en correspondance avec des ontologies OWL. Cela enrichirait BioMoby d'une hiérarchie de types de données dont la correspondance avec des concepts ontologiques est déjà établie. En utilisant BioXSD (Kalas et al. 2010) qui est un

format d'échange de données biologiques annoté en SAWSDL, BioMoby augmenterait ses capacités sémantiques.

La plateforme BioSemantic est fonctionnelle mais bénéficierait d'évolutions. Une perspective à court terme sera la possibilité d'ajouter automatiquement un SWS dans l'annuaire BioCatalogue. Cette étape doit actuellement être réalisée via des interfaces graphiques en raison de l'absence d'une librairie autorisant l'enregistrement automatique. Lorsque cette étape sera réalisée, la seule étape manuelle restera l'annotation sémantique des vues RDF.

Actuellement l'annotation sémantique des vues RDF est réalisée manuellement dans un éditeur de texte. Comme nous l'avons précisé dans les perspectives (section 9.2.2), cette étape peut être dans un premier temps améliorée par une interface graphique. Une perspective à plus long terme serait l'utilisation d'un outil de mapping automatique de schéma (i.e.WebSmatch).

Les requêtes SPARQL créées automatiquement, sont obtenues par la recherche du plus court chemin reliant deux nœuds d'un graphe. Cette approche permet de créer des requêtes prenant un paramètre en entrée et renvoyant un seul paramètre en sortie. Le développement d'un algorithme permettant de trouver le plus court chemin reliant plusieurs nœuds d'entrée à plusieurs nœuds de sortie permettrait de créer de requêtes SPARQL prenant plusieurs paramètres d'entrée et renvoyant plusieurs paramètres de sortie.

Enfin, le langage de mapping entre bases de données relationnelles et ontologies R2RML (S. Das et al. 2011) (RDB To RDF Mapping Language) a été proposé au W3C le 20 septembre 2011. R2RML répond aux mêmes besoins que le langage D2RQ mais peut potentiellement devenir un standard recommandé par le W3C. Si R2RML est validé comme standard, il sera intéressant de modifier légèrement notre implémentation pour prendre en entrée non plus une vue RDF D2RQ mais une vue RDF R2RML.

Bibliographie

Alalwan, N., Zedan, H. & Siewe, F., 2009. Generating OWL Ontology for Database Integration. Dans *Third International Conference on Advances in Semantic Processing, 2009. SEMAPRO '09*. Third International Conference on Advances in Semantic Processing, 2009. SEMAPRO '09. IEEE, p. 22-31.

An, Y., Borgida, A. & Mylopoulos, J., 2006. Discovering the Semantics of Relational Tables Through Mappings. *LNCS 4244 - JOURNAL ON DATA SEMANTICS VII*, p.1--32.

Angles, R. & Gutierrez, C., 2008. Survey of graph database models. *ACM Computing Surveys*, 40, p.1-39.

Antezana, E., Blondé, W., et al., 2009. BioGateway: a semantic systems biology tool for the life sciences. *BMC bioinformatics*, 10 Suppl 10(Suppl 10), p.S11.

Antezana, E., Kuiper, M. & Mironov, V., 2009. Biological knowledge management: the emerging role of the Semantic Web technologies. *Briefings in Bioinformatics*, 10(4), p.392 -407.

Ashburner, Michael et al., 2000. Gene Ontology: tool for the unification of biology. *Nature genetics*, 25(1), p.25-29.

Astrova, I., 2009. Rules for Mapping SQL Relational Databases to OWL Ontologies. Dans *Metadata and Semantics*. Springer US, p. 415-424. Available at: http://dx.doi.org/10.1007/978-0-387-77745-0_40 [Consulté septembre 21, 2011].

Baader, F. et al., 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*, Cambridge University Press.

Bajda-Pawlikowski, K., 2008. Querying RDF data stored in DBMS: SPARQL to SQL Conversion. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.150.8592.

Barrasa, J., Corcho, Ó. & Gómez-pérez, A., 2004. R2O, an Extensible and Semantically based Database-to-Ontology Mapping Language. *IN IN PROCEEDINGS OF THE 2ND WORKSHOP ON SEMANTIC WEB AND DATABASES(SWDB2004*, 3372, p.1069--1070.

Barrasa, J., Rodriguez & Gómez-Pérez, A., 2006. Upgrading relational legacy data to the semantic web. Dans ACM Press, p. 1069. Available at: http://dl.acm.org/citation.cfm?id=1136019 [Consulté septembre 19, 2011].

Baxevanis, A.D., 2001. The Molecular Biology Database Collection: an updated compilation of biological database resources. *Nucleic Acids Research*, 29(1), p.1-10.

Bechhofer, Sean et al., 2004. OWL Web Ontology Language Reference. Available at: http://www.w3.org/TR/owl-ref/ [Consulté octobre 4, 2011].

Beckett, Dave & McBride, B., 2004. RDF/XML Syntax Specification (Revised). *W3C*. Available at: http://www.w3.org/TR/REC-rdf-syntax/ [Consulté juillet 27, 2010].

Beckett, Dave & Broekstra, J., 2008. SPARQL Query Results XML Format. Available at: http://www.w3.org/TR/rdf-sparql-XMLres/ [Consulté octobre 3, 2011].

Beckett, Dave & Grant, J., 2003. SWAD-Europe Deliverable 10.2: Mapping Semantic Web Data with RDBMSes. Available at: http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/.

Beckett, David & Berners-Lee, T., 2011. Turtle - Terse RDF Triple Language. Available at: http://www.w3.org/TeamSubmission/turtle/ [Consulté septembre 30, 2011].

Belleau, F. et al., 2008. Bio2RDF: towards a mashup to build bioinformatics knowledge systems. *Journal of Biomedical Informatics*, 41(5), p.706-716.

Bellman, R., 1958. On a routing problem. *Quarterly of Applied Mathematics*, p.87–90.

Bergamaschi, S, 2001. Semantic integration of heterogeneous information sources. *Data & Knowledge Engineering*, 36, p.215-249.

Bergamaschi, Sonia et al., 2005. Building a Tourism Information Provider With the Momis System. *Information Technology Tourism*, 7(3), p.221-238.

Bergman, M., 2010. A New Methodology for Building Lightweight, Domain Ontologies. *AI3:::Adaptive Information*. Available at: http://www.mkbergman.com/908/a-new-methodology-for-building-lightweight-domain-ontologies/ [Consulté octobre 4, 2011].

Berners-Lee, T., 2006. Linked Data - Design Issues. Available at: http://www.w3.org/DesignIssues/LinkedData.html [Consulté octobre 2, 2011].

Berners-Lee, T., 1998. Semantic Web Road map. Available at: http://www.w3.org/DesignIssues/Semantic.html.

Berners-Lee, T., 1994. Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web. Available at: http://tools.ietf.org/html/rfc1630 [Consulté septembre 28, 2011].

Berners-Lee, T. & Connolly, D., 2011. Notation3 (N3): A readable RDF syntax. Available at: http://www.w3.org/TeamSubmission/n3/ [Consulté septembre 30, 2011].

Berners-lee, T. et al., 2006. Tabulator: Exploring and analyzing linked data on the semantic web. *IN PROCEEDINGS OF THE 3RD INTERNATIONAL SEMANTIC WEB USER INTERACTION WORKSHOP*. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.97.950.

Berners-Lee, T., Hendler, James & Lassila, O., 2001. The Semantic Web. *Scientific American*.

Bhagat, J., Tanoh, F. & al., 2010. BioCatalogue: a universal catalogue of web services for the life sciences. *Nucleic Acids Research*, 38(Web Server), p.W689-W694.

Bizer, Chris & Andreas Schultz, 2009. Berlin SPARQL Benchmark Results - 2009. Available at: http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V3/#comparison [Consulté octobre 2, 2011].

Bizer, Chris & Cyganiak, R., 2006. D2R Server – Publishing Relational Databases on the Semantic Web. Available at: http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/ [Consulté octobre 3, 2011].

Bizer, Chris & Schultz, A., 2011a. Berlin SPARQL Benchmark. Available at: http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/spec/ [Consulté octobre 2, 2011].

Bizer, Chris & Schultz, A., 2011b. Berlin SPARQL Benchmark Results - 02/22/2011. Available at: http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/results/V6/#comparison [Consulté octobre 2, 2011].

Bizer, Chris, Cyganiak, R. & Heath, T., 2007. How to publish Linked Data on the Web. Available at: http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/ [Consulté octobre 2, 2011].

Bizer, Chris & Gauß, T., 2007. Disco Hyperdata Browser. Available at: http://www4.wiwiss.fu-berlin.de/bizer/ng4j/disco/ [Consulté octobre 18, 2011].

Bizer, Christian, 2003. D2R MAP - A Database to RDF Mapping Language. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.862.

Bizer, Christian, 2004. D2RQ - treating non-RDF databases as virtual RDF graphs. *IN PROCEEDINGS OF THE 3RD INTERNATIONAL SEMANTIC WEB CONFERENCE (ISWC2004*. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.126.2314 [Consulté mars 17, 2009].

Bizer, Christian & Schultz, A., 2009. The Berlin SPARQL Benchmark. *International Journal on Semantic Web and Information Systems*, 5, p.1-24.

Boley, H. & Kifer, M., 2010. RIF Basic Logic Dialect. Available at: http://www.w3.org/TR/2010/REC-rif-bld-20100622/ [Consulté septembre 28, 2011].

Booth, D. et al., 2004. Web Services Architecture. Available at: http://www.w3.org/TR/ws-arch/ [Consulté octobre 7, 2011].

Borgida, A., Lenzerini, M. & Rosati, R., 2003. Description Logics for Databases. Dans *The Description Logic Handbook*. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P. Patel-Schneider, p. 462--484.

Bray, T. et al., 2008. Extensible Markup Language (XML) 1.0 (Fifth Edition). Available at: http://www.w3.org/TR/REC-xml/ [Consulté septembre 28, 2011].

Brickley, D. & Guha, R.V., 2004. RDF Vocabulary Description Language 1.0: RDF Schema. Available at: http://www.w3.org/TR/rdf-schema/ [Consulté septembre 30, 2011].

Brickley, D. & Miller, L., 2010. FOAF Vocabulary Specification. Available at: http://xmlns.com/foaf/spec/ [Consulté septembre 30, 2011].

Broekstra, J., Kampman, A. & Van Harmelen, F., 2001. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. *SEMANTICS FOR THE WWW*. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.7561.

Bruijn, J. de et al., 2005. Web Service Modeling Ontology (WSMO). Available at: http://www.w3.org/Submission/WSMO/ [Consulté octobre 10, 2011].

Bruskiewich, Richard, Senger, Martin, Davenport, G., Ruiz, M., Rouard, M., Hazekamp, T., et al., 2008. The Generation Challenge Programme Platform: Semantic Standards and Workbench for Crop Science. *International Journal of Plant Genomics*, 2008, p.369601.

Bruskiewich, Richard, Davenport, G. & al., 2006. Generation Challenge Programme (GCP): standards for crop data. *Omics: A Journal of Integrative Biology*, 10(2), p.215-219.

Bruskiewich, Richard, Senger, Martin, Davenport, G., Ruiz, M., Rouard, M. & al, 2008. The Generation Challenge Programme Platform: Semantic Standards and Workbench for Crop Science. *International Journal of Plant Genomics*. Available at: http://www.hindawi.com/GetArticle.aspx?doi=10.1155/2008/369601&e=cta.

Calvanese, D. et al., 2011. The MASTRO system for ontology-based data access. *Semantic Web*, 2(1), p.43-53.

Chang, K.C.-C. et al., 2004. Structured databases on the web. *ACM SIGMOD Record*, 33, p.61.

Chebotko, A., Lu, S. & Fotouhi, F., 2009. Semantics preserving SPARQL-to-SQL translation. *Data & Knowledge Engineering*, 68, p.973-1000.

Chen, P.P.-shan, 1976. The Entity-Relationship Model: Toward a Unified View of Data. *ACM TRANSACTIONS ON DATABASE SYSTEMS*, 1, p.9--36.

Cheung, K.-H. et al., 2005. YeastHub: a semantic web use case for integrating data in the life sciences domain. *Bioinformatics*, 21, p.i85-i96.

Chinnici, R. et al., 2007. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. Available at: http://www.w3.org/TR/wsdl20/ [Consulté octobre 8, 2011].

Clark, K.G., Feigenbaum, L. & Torres, E., 2008. SPARQL Protocol for RDF. Available at: http://www.w3.org/TR/2008/REC-rdf-sparql-protocol-20080115/ [Consulté octobre 3, 2011].

Clement, L. et al., 2004. UDDI Version 3.0.2. Available at: http://www.uddi.org/pubs/uddi_v3.htm [Consulté octobre 9, 2011].

Codd, E.F, 1971. Further Normalization of the Data Base Relational Model. *IBM Research Report*.

Codd, E. F., 1970. A relational model of data for large shared data banks. *Communications of the ACM*, 13, p.377-387.

Codd, Edgar F., 1974. *Recent Investigations into Relational Data Base Systems*, IBM.

Cohen-Boulakia, S., 2005. Intégration de données biologiques: Sélection de sources centrée sur l'utilisateur. PhD thesis, Univ. de Paris Sud Orsay.

Comyn-Wattiau, I. & Akoka, J., 1996. Reverse Engi- neering of Relational Database Physical Schemas. Dans *Proceedings of the Fifteenth International Conference on the Entity-Relationship Approach*. Cottbus, Germany, p. 372-391.

Cook, A.F., IV & Wenk, C., 2008. Min-Link Shortest Path Maps and Fréchet Distance. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.163.8885.

Corby, O., Dieng-kuntz, R. & Faron-zucker, C., 2004. Querying the Semantic Web with the CORESE search engine. , p.705--709.

Cornell, M. et al., 2001. GIMS - A Data Warehouse for Storage and Analysis of Genome Sequence and Functional Data. *IN PROC. 2ND IEEE SYMPOSIUM ON BIOINFORMATICS AND BIOENGINEERING (BIBE*, p.15--22.

Côté, R.G. et al., 2006. The Ontology Lookup Service, a lightweight cross-platform tool for controlled vocabulary queries. *BMC Bioinformatics*, 7, p.97-97.

Covitz, P.A. et al., 2003. caCORE: a common infrastructure for cancer informatics. *Bioinformatics*, 19(18), p.2404—2412.

Cullot, N., Ghawi, R. & Y'etongnon, K., 2007. DB2OWL : A Tool for Automatic Database-to-Ontology Mapping. Dans *SEBD*. p. 494, 491. Available at: http://dblp.uni-trier.de/rec/bibtex/conf/sebd/CullotGY07 [Consulté février 12, 2009].

Curino, C. et al., 2009. Accessing and Documenting Relational Databases through OWL Ontologies. Dans T. Andreasen et al., éd. *Flexible Query Answering Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, p. 431-442. Available at: http://www.springerlink.com/content/v00q30045172v2n4/ [Consulté septembre 19, 2011].

Cyga, R., 2005. A relational algebra for SPARQL.

Cyganiak, R. & Jentzsch, A., 2011. Linking Open Data cloud diagram. Available at: http://lod-cloud.net/.

Das, S., Sundara, S. & Cyganiak, R., 2011. R2RML: RDB to RDF Mapping Language. Available at: http://www.w3.org/TR/r2rml/ [Consulté octobre 20, 2011].

Date, C.J., 1995. An introduction to database systems. Dans Addison-Wesley Pub. Co., p. 290.

Davidson, S.B. et al., 2001. K2/Kleisli and GUS: Experiments in integrated access to genomic data sources. *IBM Systems Journal*, 40, p.512-531.

Davidson, S.B., Overton, C. & Buneman, P., 1995. Challenges in Integrating Biological Data Sources. *JOURNAL OF COMPUTATIONAL BIOLOGY*, 2, p.557--572.

Davis, K.H. & Arora, A.K., 1987. Converting A Relational Database Model into an Entity-Relationship Model. Dans *Proceedings of the Sixth International Conference on Entity-Relationship Approach*.

Day-Richter, J. et al., 2007. OBO-Edit--an ontology editor for biologists. *Bioinformatics (Oxford, England)*, 23(16), p.2198-2200.

Deus, H.F. et al., 2010. Exposing the cancer genome atlas as a SPARQL endpoint. *Journal of Biomedical Informatics*, 43(6), p.998-1008.

DiBernardo, M., Pottinger, R. & Wilkinson, M., 2008. Semi-automatic web service composition for the life sciences using the BioMoby semantic web framework. *Journal of Biomedical Informatics*, 41, p.837-847.

Dijkstra, E., 1959. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1), p.269-271.

Domingos, P., 2003. Prospects and challenges for multi-relational data mining. *ACM SIGKDD Explorations Newsletter*, 5, p.80.

Dou, D. & LePendu, P., 2006. Ontology-based integration for relational databases. Dans ACM Press, p. 461. Available at: http://dl.acm.org/citation.cfm?id=1141387&dl=ACM&coll=DL&CFID=44687710&CFTOKEN=82526049 [Consulté septembre 18, 2011].

Eilbeck, K et al., 2005. The Sequence Ontology: a tool for the unification of genome annotations. *Genome Biology*, 6(5), p.R44.

Elliott, B. et al., 2009. A complete translation from SPARQL into efficient SQL. Dans ACM Press, p. 31. Available at: http://dl.acm.org/citation.cfm?id=1620437 [Consulté septembre 21, 2011].

Elmasri, R., 2006. *Fundamentals of database systems* 5e éd., Boston: Pearson.

Eric Sayers, D.W., 2004. Building Customized Data Pipelines Using the Entrez Programming Utilities (eUtils). Available at: http://www.ncbi.nlm.nih.gov/books/NBK1058/ [Consulté octobre 10, 2011].

Erik Christensen et al., 2001. Web Services Description Language (WSDL). Available at: http://www.w3.org/TR/wsdl [Consulté octobre 9, 2011].

Erling, O. & Mikhailov, I., 2006. Mapping Relational Data to RDF in Virtuoso. Available at: http://virtuoso.openlinksw.com/wiki/main/Main/VOSSQLRDF [Consulté septembre 7, 2011].

Etzold, T., Ulyanov, A. & Argos, P., 1996. SRS: information retrieval system for molecular biology data banks. *Methods in Enzymology*, 266, p.114-128.

Farrell, J. & Lausen, H., 2007. Semantic Annotations for WSDL and XML Schema. Available at: http://www.w3.org/TR/sawsdl/ [Consulté octobre 10, 2011].

Fensel, D. & Bussler, C., 2002. The Web Service Modeling Framework WSMF (Extended Abstract). Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.7563.

Fielding, R.T., 2000. Architectural Styles and the Design of Network-based Software Architectures. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.2433.

Fikes, R. & Kehler, T., 1985. The role of frame-based representation in reasoning. *Communications of the ACM*, 28, p.904-920.

Florescu, D., Levy, A. & Mendelzon, A., 1998. Database techniques for the World-Wide Web. *ACM SIGMOD Record*, 27, p.59-74.

Floyd, R.W., 1962. Algorithm 97: Shortest path. *Communications of the ACM*, 5, p.345.

Galperin, MY, 2007. The Molecular Biology Database Collection: 2007 update. *Nucleic Acids Res*, 35(Database issue). Available at: http://view.ncbi.nlm.nih.gov/pubmed/17148484 [Consulté septembre 23, 2011].

Galperin, Michael & Cochrane, G., 2011. The 2011 Nucleic Acids Research Database Issue and the online Molecular Biology Database Collection. , 39(Database issue), p.D1-D6.

Garlik, 2009. 4store - Scalable RDF storage. Available at: http://4store.org/ [Consulté octobre 18, 2011].

Gessler, Damian, Schiltz, G. & al., 2009. SSWAP: A Simple Semantic Web Architecture and Protocol for semantic web services. *BMC Bioinformatics*, 10(1), p.309.

Gessler, D.D.G., Schiltz, G.S., et al., 2009. SSWAP: A Simple Semantic Web Architecture and Protocol for semantic web services. *BMC Bioinformatics*, 10, p.309.

De Giovanni, R. et al., 2010. TAPIR - TDWG Access Protocol for Information Retrieval, version 1.0. Available at: http://www.tdwg.org/dav/subgroups/tapir/1.0/docs/tdwg_tapir_specification_2010-05-05.htm [Consulté octobre 11, 2011].

GMOD, 2007. Generic Model Organism Database. Available at: http://gmod.org/ [Consulté septembre 26, 2011].

Goble, Carole & Stevens, Robert, 2008. State of the nation in data integration for bioinformatics. *Journal of Biomedical Informatics*, 41(5), p.687–93.

Goble, Carole & De Roure, D., 2008. Curating Scientific Web Services and Workflows. Available at: http://eprints.ecs.soton.ac.uk/16561/ [Consulté octobre 10, 2011].

Goff, S.A. et al., 2011. The iPlant collaborative: cyberinfrastructure for plant biology. *Frontiers in Plant Genetics and Genomics*, 2, p.34.

Golbreich, C. & Horrocks, I., 2007. The OBO to OWL mapping, GO to OWL 1.1. *IN PROCEEDINGS OF THE OWLED 2007 WORKSHOP ON OWL: EXPERIENCES AND DIRECTIONS*. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.168.2758.

Golbreich, C. et al., 2008. OBO and OWL: Leveraging Semantic Web Technologies for the Life Sciences. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.97.5075.

Gray, P., 1984. *Logic, algebra, and databases*, Chichester West Sussex England; New York: E. Horwood;Distributor Halsted Press.

Grosof, B.N. et al., 2003. Description logic programs: combining logic programs with description logic. Dans ACM Press, p. 48. Available at: http://dl.acm.org/citation.cfm?id=775160 [Consulté septembre 28, 2011].

Gruber, T.R., 1993. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. *IN FORMAL ONTOLOGY IN CONCEPTUAL ANALYSIS AND KNOWLEDGE REPRESENTATION, KLUWER ACADEMIC PUBLISHERS, IN PRESS. SUBSTANTIAL REVISION OF PAPER PRESENTED AT THE INTERNATIONAL WORKSHOP ON FORMAL ONTOLOGY*. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.122.3207.

Gudgin, M. et al., 2007. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Available at: http://www.w3.org/TR/soap12-part1/ [Consulté octobre 24, 2011].

Guérin, E., 2005. Integration de donnees pour l'analyse de transcriptome: mise en oeuvre par l'entrepot GEDAW (Gene Expression Data Warehouse). PhD thesis, Univ. de Rennes 1

Guo, Y, Pan, Z. & Heflin, J., 2005. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3), p.158-182.

Haarslev, V. & Möller, R., 2001. Description of the RACER System and its Applications. , p.132--141.

Haas, L.M. et al., 2001. DiscoveryLink: a system for integrated access to life sciences data sources. *IBM Syst. J.*, 40(2), p.489—511.

Hakala, J., 2000. Dublin Core Metadata Initiative. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.21.5539.

Halevy, A.Y., 2001. Answering queries using views: A survey. *VLDB Journal: Very Large Data Bases*, 10(4), p.270—294.

Hamill, R. & Martin, N., 2004. Database Support for Path Query Functions. Dans H. Williams & L. MacKinnon, éd. *Key Technologies for Data Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, p. 84-99. Available at: http://www.springerlink.com/content/c82u7w6p2gxqlfea/ [Consulté octobre 17, 2011].

Harris, S. & Gibbins, N., 2003. 3store: Efficient Bulk RDF Storage. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.165.8459.

Hayes, P., 2004. RDF Semantics. Available at: http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ [Consulté septembre 29, 2011].

Hedeler, C. et al., 2007. e-Fungi: a data resource for comparative analysis of fungal genomes. *BMC Genomics*, 8, p.426.

Hernandez, T. & Kambhampati, S., 2004. Integration of Biological Sources: Current Systems and Challenges Ahead. *SIGMOD RECORD*, 33, p.51--60.

Hodges, W., 2001. Classical logic I— first-order logic. Dans *The Blackwell Guide to Philosophical Logic*. p. 9-32.

Horridge, M. & Bechhofer, Sean, 2009. The OWL API: A Java API for Working with OWL 2 Ontologies. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.163.7035.

Horrocks, I. et al., 2004. The instance store: Description logic reasoning with large numbers of individuals. *IN 'INTERNATIONAL WORKSHOP ON DESCRIPTION LOGICS (DL 2004*, p.31--40.

Huang, X. et al., 2010. Genome-wide association studies of 14 agronomic traits in rice landraces. *Nature Genetics*, 42(11), p.961-967.

Hull, Duncan, Stevens, Robert & Lord, P., 2005. Describing web services for user-oriented retrieval. *IN PROC. OF W3C WORKSHOP ON FRAMEWORKS FOR SEMANTICS IN WEB SERVICES*, p.9--10.

Hull, D. et al., 2006. Taverna: a tool for building and running workflows of services. *Nucleic Acids Research*, 34, p.W729-W732.

Inmon, W.H., 2005. *Building the data warehouse*, Wiley.

Jaiswal, P. et al., 2005. Plant Ontology (PO): a Controlled Vocabulary of Plant Structures and Growth Stages. *Comparative and Functional Genomics*, 6(7-8), p.388-397.

Kalas, M. et al., 2010. BioXSD: the common data-exchange format for everyday bioinformatics web services. *Bioinformatics (Oxford, England)*, 26(18), p.i540-546.

Kanehisa, M. et al., 2002. The KEGG databases at GenomeNet. *Nucleic Acids Research*, 30(1), p.42-46.

Kawas, E., Senger, Martin & Wilkinson, M., 2006. BioMoby extensions to the Taverna workflow management and enactment software. *BMC Bioinformatics*, 7(1), p.523.

Kersey, P. et al., 2005. Integr8 and Genome Reviews: integrated views of complete genomes and proteomes. *Nucleic Acids Research*, 33(Database Issue), p.D297-D302.

Kiringa, I., 2009. Le Modèle Entité-Relation. Available at: http://www.site.uottawa.ca/~kiringa/courses09/csi2532/ch2_er_csi2532-09.ppt.

Klyne, G. & Carroll, J., 2004. Resource Description Framework (RDF): Concepts and Abstract Data Model. Available at: http://www.w3.org/TR/2002/WD-rdf-concepts-20020829/#section-URIspaces [Consulté septembre 28, 2011].

Knublauch, H. et al., 2004. The Protégé OWL plugin: An open development environment for semantic web applications. , p.229--243.

Koehler, J. et al., 2005. Linking experimental results, biological networks and sequence analysis methods using Ontologies and Generalised Data Structures. *In Silico Biology*, 5(1), p.33-44.

Konstantinou, N., Spanos, D.-E. & Mitrou, N., 2008. ONTOLOGY AND DATABASE MAPPING: A SURVEY OF CURRENT IMPLEMENTATIONS AND FUTURE DIRECTIONS. *Journal of Web Engineering*, 7(1), p.001-024.

de Laborda, C.P. et al., 2005. Relational.OWL - A Data and Schema Representation Format Based on OWL. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.6179.

Lam, H.Y.K. et al., 2006. Using Web Ontology Language to Integrate Heterogeneous Databases in the Neurosciences. , 2006, p.464-468.

Larmande, P., 2007. Mutualiser et partager, un défi pour la génomique fonctionnelle végétale. PhD thesis, Univ. Montpellier 2

Lee, T.J. et al., 2006. BioWarehouse: a bioinformatics database warehouse toolkit. *BMC Bioinformatics*, 7, p.170.

Lenzerini, M., 2002. Data integration: a theoretical perspective. Dans *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM Press, p. 246, 233. Available at: http://dx.doi.org/10.1145/543613.543644 [Consulté avril 6, 2009].

Levy, A.Y., 1999. Combining Artificial Intelligence and Databases for Data Integration. Dans M. J. Wooldridge & M. Veloso, éd. *Artificial Intelligence Today*. Berlin, Heidelberg: Springer Berlin Heidelberg, p. 249-268. Available at: http://www.springerlink.com/content/147463283t666lu4/ [Consulté septembre 26, 2011].

Li, Man, Du, X. & Wang, S., 2005. A Semi-automatic Ontology Acquisition Method for the Semantic Web. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.2302.

López-garcía, P., Mena, E. & Bermúdez, J., 2009. SOME COMMON PITFALLS IN THE DESIGN OF ONTOLOGY-DRIVEN INFORMATION SYSTEMS. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.149.4489.

Lu, J. et al., 2008. An Effective SPARQL Support over Relational Databases. Dans V. Christophides, M. Collard, & C. Gutierrez, éd. *Semantic Web, Ontologies and Databases*. Berlin, Heidelberg: Springer Berlin Heidelberg, p. 57-76. Available at: http://dl.acm.org/citation.cfm?id=1423679&coll=GUIDE&dl=GUIDE [Consulté septembre 19, 2011].

Martin, D. et al., 2004. OWL-S: Semantic Markup for Web Services. Available at: http://www.w3.org/Submission/OWL-S/ [Consulté octobre 10, 2011].

Martin, S., Hohman, M.M. & Liefeld, T., 2005. The impact of Life Science Identifier on informatics data. *Drug Discovery Today*, 10(22), p.1566-1572.

McGuinness, D.L. & Harmelen, F. van, 2004. OWL Web Ontology Language Overview. Available at: http://www.w3.org/TR/owl-features/ [Consulté septembre 28, 2011].

Mcwilliam, H. et al., 2009. Web services at the European Bioinformatics Institute-2009. *Nucleic Acids Research*. Available at: http://nar.oxfordjournals.org/content/early/2009/05/12/nar.gkp302.abstract [Consulté octobre 10, 2011].

Merali, Z. & Giles, J., 2005. Databases in peril. *Nature*, 435(7045), p.1010-1011.

Miles, A. et al., 2010. OpenFlyData: An exemplar data web integrating gene expression data on the fruit fly Drosophila melanogaster. *Journal of Biomedical Informatics*. Available at: http://linkinghub.elsevier.com/retrieve/pii/S1532046410000511.

Motik, B., Patel-Schneider, P.F. & Parsia, B., 2009. OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax. Available at: http://www.w3.org/TR/owl2-syntax/ [Consulté octobre 4, 2011].

Mungall, Chris & Ireland, A., 2010. The OBO Flat File Format Guide, version 1.4. Available at: http://www.geneontology.org/GO.format.obo-1_4.shtml#S.0 [Consulté octobre 4, 2011].

Mungall, C.J., Emmert, D.B. & Consortium, F., 2007. A Chado case study: an ontology-based modular schema for representing genome-associated biological information. *Bioinformatics*, 23(13), p.i337—i346.

Nelson, R.T. et al., 2010. Applications and methods utilizing the Simple Semantic Web Architecture and Protocol (SSWAP) for bioinformatics resource discovery and disparate data and service integration. *BioData Mining*, 3(1), p.3.

Nolin, M.-A. et al., 2010. Bio2RDF: Convert, Provide And Reuse. *Nature Precedings*. Available at: http://precedings.nature.com/documents/5060/version/1 [Consulté octobre 3, 2011].

Noy, N. F et al., 2001. Creating Semantic Web contents with Protege-2000. *IEEE Intelligent Systems*, 16(2), p.60- 71.

Nyulas, C. & Tu, S., 2007. DataMaster – a Plug-in for Importing Schemas and Data from Relational Databases into Protégé. *IN PROCEEDINGS OF 10 TH INTERNATIONAL PROTÉGÉ CONFERENCE*. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.170.497.

Object Management Group, 2002. MOF 2.0 Query/Views/Transformations RFP. Available at: http://www.omg.org/cgi-bin/doc?ad/2002-4-10 [Consulté juin 12, 2011].

OBO Foundry, 2008. OBO Foundry Principles - OBOFoundry. Available at: http://obofoundry.org/wiki/index.php/OBO_Foundry_Principles_2008 [Consulté octobre 4, 2011].

Oinn, T. et al., 2004. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17), p.3054, 3045.

Oladele, R., 2008. INTEGRATION OF BIOLOGICAL DATA SOURCES: A SURVEY.

OpenLink Software, 2009. Open Virtuoso. Available at: http://virtuoso.openlinksw.com/ [Consulté octobre 6, 2011].

Paslaru, E., Simperl, B. & Tempich, C., 2006. Ontology Engineering: A Reality Check. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.72.5445.

Pasquier, C., 2008. Biological data integration using Semantic Web technologies. *Biochimie*, 90(4), p.584-94.

Pierre-Alain Muller & Nathalie Gaertner, 2000. *Modélisation objet avec UML* Deuxième édition., Eyrolles.

Pillai, S. et al., 2005. SOAP-based services provided by the European Bioinformatics Institute. *Nucleic Acids Research*, 33(Web Server issue), p.W25-W28.

Poggi, A., Rodriguez-muro, M. & Ruzzi, M., 2008. Ontology-based database access with DIG-Mastro and the OBDA Plugin for Protégé. *PATEL-SCHNEIDER (EDS.), PROC. OF THE 4TH INT. WORKSHOP ON OWL: EXPERIENCES AND DIRECTIONS (OWLED 2008 DC*. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.159.2328.

ProgrammableWeb.com, 2010. ProgrammableWeb - Mashups, APIs, and the Web as Platform. Available at: http://www.programmableweb.com/ [Consulté septembre 26, 2011].

Prud'hommeau, E. & Seaborne, A., 2008. SPARQL Query Language for RDF. *W3C*. Available at: http://www.w3.org/TR/rdf-sparql-query/ [Consulté juillet 27, 2010].

Prud'hommeaux, E. & Seaborne, A., 2008. SPARQL Query Language for RDF. Available at: http://www.w3.org/TR/rdf-sparql-query/ [Consulté octobre 3, 2011].

Prud'hommeaux, E. & Seaborne, A., 2004. SPARQL Query Language for RDF. Available at: http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/ [Consulté octobre 3, 2011].

R. Fielding et al., 1999. Hypertext Transfer Protocol -- HTTP/1.1. Available at: http://tools.ietf.org/html/rfc2616 [Consulté octobre 27, 2011].

Ramakrishnan, R. & Gehrke, J., 2002. *Database Management Systems* third ed., New York City.

RDF Working Group, 2004. RDF - Semantic Web Standards. Available at: http://www.w3.org/RDF/ [Consulté septembre 28, 2011].

Redaschi, N. & Consortium, U., 2009. UniProt in RDF: Tackling Data Integration and Distributed Annotation with the Semantic Web. *Nature Precedings*. Available at: http://precedings.nature.com/documents/3193/version/1 [Consulté octobre 3, 2011].

Reynaud, C. & Safar, B., 2009. Construction automatique d'adaptateurs guidée par une ontologie pour l'intégration de sources et de données XML. Available at: http://hal.inria.fr/inria-00432426/en/ [Consulté septembre 27, 2011].

Rother, K. et al., 2004. COLUMBA: Multidimensional Data Integration of Protein Annotations. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.9641.

Ruiz, M. et al., 2004. TropGENE-DB, a multi-tropical crop information system. *Nucleic Acids Research*, 32(Database issue), p.D364-367.

Ruttenberg, A. et al., 2009. Life sciences on the Semantic Web: the Neurocommons and beyond. *Briefings in Bioinformatics*, 10(2), p.193-204.

Sahoo, S.S. et al., 2007. SemBOWSER - adding Semantics to biological Web services registry' in Semantic Web. Dans *Revolutionizing Knowledge Discovery in the Life Sciences*. Christopher J. O. Baker and Kei-Hoi Cheung. New York: Springer, p. 317-340.

Schmolze, J.G., Beranek, B. & Inc, N., 1985. An overview of the KL-ONE knowledge representation system. *COGNITIVE SCIENCE*, 9, p.171--216.

Schober, D. et al., 2009. Survey-based naming conventions for use in OBO Foundry ontology development. *BMC Bioinformatics*, 10(1), p.125.

Schuler, G.D. et al., 1996. Entrez: molecular biology database and retrieval system. *Methods in Enzymology*, 266, p.141-162.

Senger, Martin et al., 2009. Soaplab: Open Source Web Services Framework for Bioinformatics Programs. Dans Bosc.

Shah, S.P. et al., 2005. Atlas - a data warehouse for integrative bioinformatics. *BMC Bioinformatics*, 6, p.34.

Sheth, A.P. & Larson, J.A., 1990. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM COMPUTING SURVEYS*, 22, p.183--236.

Shrestha, R. et al., 2010. Multifunctional crop trait ontology for breeders' data: field book, annotation, data discovery and semantic enrichment of the literature. *AoB Plants*, 2010, p.plq008-plq008.

Smith, B. et al., 2007. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. *Nat Biotech*, 25(11), p.1251-1255.

Sowa, J.E. & Shapiro, S.C., 2006. Knowledge Representation: Logical, Philosophical, and Computational Foundations Computational Foundations by John F. Sowa (Book

Review). Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.3517.

Sowa, J.F., 1984. Conceptual graphs. *INFORMATION PROCESSING IN MIND AND MACHINE*, p.39--44.

Spanos, D.-E., Stavrou, P. & Mitrou, N., 2011. Bringing Relational Databases into the Semantic Web: A Survey. *IOS Press*.

Stein, Lincoln, 2002. Creating a bioinformatics nation. *Nature*, 417(6885), p.119-120.

Stevens, R.D. et al., 2000. TAMBIS: transparent access to multiple bioinformatics information sources. *BIOINFORMATICS*, 16, p.184--186.

Summers, E. & Phipps, J., 2008. SKOS: New Dimensions in Interoperability. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.183.2282.

Taylor, C.F. et al., 2007. The minimum information about a proteomics experiment (MIAPE). *Nat Biotech*, 25(8), p.887-893.

The BioMoby Consortium, 2008. Interoperability with Moby 1.0—It's better than sharing your toothbrush! *Briefings in Bioinformatics*. Available at: http://bib.oxfordjournals.org/content/early/2008/01/31/bib.bbn003.short [Consulté mars 7, 2011].

Thor, A., Aumueller, D. & Rahm, E., 2007. Data Integration Support for Mashups. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.90.3124.

Tirmizi, S., Sequeda, J. & al., 2008. Translating SQL Applications to the Semantic Web. Dans *Database and Expert Systems Applications*. p. 450-464. Available at: http://dx.doi.org/10.1007/978-3-540-85654-2_40 [Consulté avril 22, 2009].

Törmä, S. et al., 2008. Semantic Web Services - A Survey.

Viljanen, K. et al., 2011. Combining Distributed Ontology Repositories into a Global Service.

W3C, World Wide Web Consortium (W3C). Available at: http://www.w3.org/ [Consulté septembre 28, 2011].

Wanchana, S. et al., 2008. The Generation Challenge Programme comparative plant stress-responsive gene catalogue. *Nucleic Acids Research*, 36(Database issue), p.D943–6.

Wang, X. et al., 2009. Translational integrity and continuity: Personalized biomedical data integration. *Journal of Biomedical Informatics*, 42(1), p.100-112.

Wessel, M. & Möller, R., 2005. A High Performance Semantic Web Query Answering Engine. *IN PROC. OF THE 2005 DESCRIPTION LOGIC WORKSHOP (DL 2005). CEUR ELECTRONIC WORKSHOP PROCEEDINGS, HTTP://CEUR-WS.ORG*. Available at: http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.59.6164.

Whetzel, P. et al., 2009. BioPortal: Ontologies and Integrated Data Resources at the Click of a Mouse. *Nature Precedings*. Available at: http://precedings.nature.com/documents/3868/version/1 [Consulté octobre 4, 2011].

Wiederhold, G. & Genesereth, M., 1997. The Conceptual Basis for Mediation Services. Available at: http://ilpubs.stanford.edu:8090/279/ [Consulté septembre 29, 2011].

Wiederhold, Gio, 1992. Mediators in the architecture of future information systems. *IEEE COMPUTER*, 25(3), p.38--49.

Wilkinson, M. et al., 2003. The BioMOBY Project Explores Open-Source, Simple, Extensible Protocols for Enabling Biological Database Interoperability. Available at: http://www.virtualgenomics.org/proceedings2003/VCGB_130.pdf [Consulté octobre 11, 2011].

Wilkinson, M., McCarthy, L. & al., 2010. SADI, SHARE, and the in silico scientific method. *BMC Bioinformatics*, 11(Suppl 12), p.S7.

Wilkinson, M. et al., 2005. BioMOBY successfully integrates distributed heterogeneous bioinformatics Web Services. The PlaNet exemplar case. *Plant Physiol*, 138(1), p.5—17.

Wilkinson, M.D., Vandervalk, B. & McCarthy, L., 2009. SADI Semantic Web Services – ‘cause you can’t always GET what you want! Dans SWSIP 09. p. 7. Available at: http://sadiframework.org/documentation/SADI_SWSIP09_personal.pdf [Consulté janvier 18, 2010].

Wollbrett, J., Larmande, P. & Ruiz, M., soumis_2011. Automatic generation of Semantic Web Services for relational biological databases. *BMC Bioinformatics*.

Wollbrett, J., Larmande, P. & Ruiz, M., 2009. Intégration automatique d’une ontologie de domaine dans un annuaire BioMoby. Dans JOBIM 2009.

Wollbrett, J., Larmande, P. & Ruiz, M., 2011. Towards Automatic Generation of Semantic Web Services for Relational Biological Databases. Dans RED 2011.

Wood, D., 2005. Kowari: A Platform for Semantic Web Storage and Analysis. *IN XTECH 2005 CONFERENCE*, p.05--0402.

Woods, W.A., 1975. What’s in a Link: Foundations for Semantic Networks. Dans D. G. Bobrow & A. Collins, éd. *Representation and Understanding*. Academic Press.

Xu, L. & Embley, D.W., 2007a. Combining the best of global-as-view and local-as-view for data integration. *IN INFORMATION SYSTEMS TECHNOLOGIES AND ITS APPLICATIONS - ISTA*, p.123--136.

Xu, L. & Embley, D.W., 2007b. Using Schema Mapping to Facilitate Data Integration. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.15.2941.

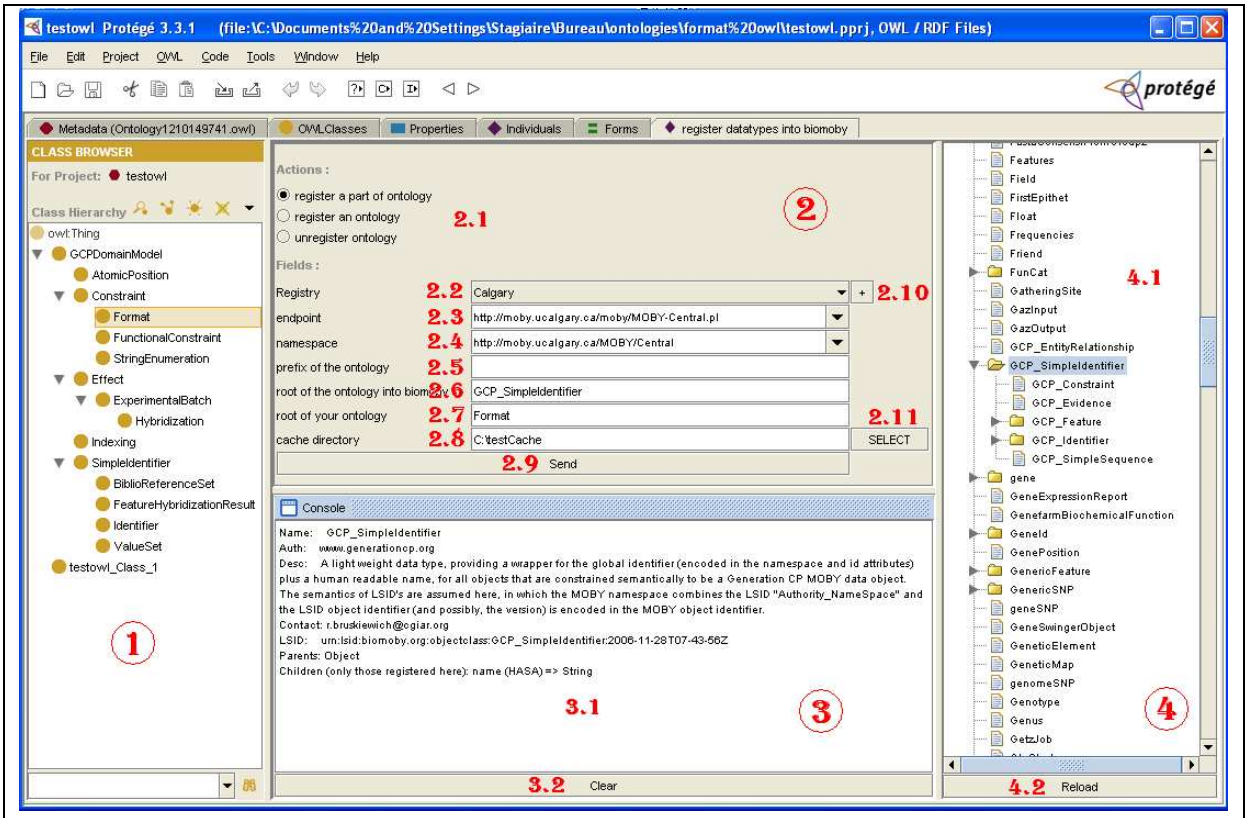
Zhang, J. et al., 2011. BioMart: a data federation framework for large collaborative projects. *Database: The Journal of Biological Databases and Curation*, 2011, p.bar038.

Zhong, W. & Sternberg, P.W., 2007. Automated data integration for developmental biological research. *Development (Cambridge, England)*, 134(18), p.3227-3238.

Annexes

Sommaire des références

Annexe A : Présentation détaillée de l'interface graphique de BioMoby Converter et détails de son d'utilisation



Légende :

1. **Arborescence de l'ontologie OWL**
2. **Zone de sélection de l'utilisateur**

2.1 Choix de l'action
2.2 Nom de l'annuaire central
2.3 Point d'entrée de l'annuaire central
2.4 Espace de nom de l'annuaire central
2.5 Préfixe de l'ontologie
2.6 Racine de l'ontologie Biomoby
2.7 Racine de l'ontologie OWL
2.8 Chemin d'accès au dossier de cache
2.9 Bouton d'envoi de l'action
2.10 Bouton pour ajouter un annuaire central
2.11 Bouton de sélection du dossier de cache

3. **Console**

3.1 Zone d'affichage de la console
3.2 Bouton pour vider la console

4. **Arborescence de l'ontologie Biomoby**

4.1 Arborescence de l'ontologie de types de données
4.2 Bouton de mise à jour de l'ontologie de types de données

Affichage et détail des panels du plugin

Nous allons maintenant détailler l'utilisation de chacun de ces panels.

1. Arborescence d'ontologie OWL

Un des panels de mon plugin consiste à visualiser l'arborescence de l'ontologie OWL qui a été chargée dans Protégé. Un panel déjà présent dans l'onglet « OWL Classes » de Protégé correspond exactement à nos attentes. En effet, il permet d'afficher l'arborescence d'une ontologie. Il permet également d'ajouter des classes à une ontologie. Enfin, en effectuant un double clic sur une classe de l'ontologie, ce panel permet d'afficher toutes les informations sur la classe (nom, description, propriétés, etc.). Nous avons également développé une méthode qui permet de récupérer le nom de la classe sélectionnée dans l'arborescence OWL. Cela permettra au plugin de récupérer automatiquement le nom de la classe dans la zone de sélection de l'utilisateur.

2. La console

La classe Console utilisée dans mon plugin est une classe récupérée sur internet[18]. Cette classe permet de rediriger les sorties standard (*System.out* et *System.err*) vers un *JTextArea*. Nous utilisons cette console pour afficher du texte dans le plugin. Cela permet à l'utilisateur de visualiser les différentes étapes d'enregistrement ou de suppression d'ontologie sous Biomoby. Cela est utile pour que l'utilisateur sache si l'action qu'il a exécuté a aboutie ou non. Nous avons créé un message d'erreur spécifique à chaque étape du processus. Cela permet à l'utilisateur de savoir exactement qu'elle étape n'a pas pu être réalisée. Il peut ainsi modifier sa manipulation pour la rendre compatible avec le plugin.

3. L'arborescence des types de données Biomoby

Le troisième panel intégré dans le plugin permet de visualiser les types de données enregistrés dans l'annuaire central sélectionné. Il s'agit d'une arborescence de types de données. Cette arborescence permet de visualiser les effets d'un enregistrement ou d'une suppression d'ontologie directement via le plugin. Cela permet de se passer totalement du logiciel Dashboard lors de la manipulation des ontologies sur un annuaire central Biomoby.

Cet arborescence possède différentes actions disponibles via un menu déroulant obtenu grâce à un clic droit de la souris. Il est possible de trier les types de données par ordre alphabétique ou par autorité. Le tri par autorité permet de regrouper tous les types de données créés par le même auteur. Il est également possible d'étendre ou de réduire tous les nœuds de l'arborescence. Une autre option disponible grâce à un clic droit est la recherche d'un type de données par son nom. Le menu permet aussi d'afficher ou non les propriétés des types de données (relations HAS et HASA). La dernière option disponible grâce au menu d'options est le rechargement des informations sur les types de données. Cette option permet d'utiliser le dossier de cache pour recharger un arbre. Cette option ne permet pas de mettre à jour un arbre mais juste de récupérer les informations présentes dans le dossier de cache. Cela est très utile et très rapide pour afficher les types de données enregistrés dans un autre annuaire central. Il suffit pour cela que l'utilisateur spécifie l'annuaire central qu'il souhaite visualiser (via la zone de sélection de l'utilisateur détaillée dans la partie suivante), puis qu'il clique sur l'option « reload ». L'arborescence des types de données présents dans cet annuaire central sera automatiquement affichée. Cette arborescence correspondra aux types de données présents dans l'annuaire central lors de la dernière mise à jour du dossier de cache. Aucune action de mise à jour du cache n'est disponible. Nous avons donc décidé d'ajouter un bouton

[18] Adresse internet: http://www.javafr.com/codes/REDIRECTION-FLUX-SYSTEM-OUT-SYSTEM-ERR-DANS-JTEXTAREA_44400.aspx

permettant la mise à jour du dossier de cache. Un clic sur ce bouton permet de se connecter à l'annuaire central Biomoby sélectionné, de vérifier les types de données différents entre le dossier de cache et l'annuaire central et de récupérer uniquement les types de données différents. Une dernière fonctionnalité a été ajoutée à cet arbre. En effet, nous voulions qu'en cliquant sur un des nœuds de l'arbre le nom du nœud apparaisse automatiquement dans la zone de sélection de l'utilisateur et que la description du type de données apparaisse dans la console. Ce clic induit également une recherche des informations sur le type de données sélectionné et retourne toutes les informations dans la console du plugin.

4. La zone de sélection de l'utilisateur

Il s'agit de la partie la plus importante du plugin. Celle qui permet de communiquer avec tous les autres panels ainsi qu'avec les classes permettant le parcours du fichier OWL, le tri des types de données et l'ajout ou la suppression d'ontologies. Il s'agit du panel où l'utilisateur choisit les actions qu'il souhaite réaliser avec les paramètres appropriés.

La première partie du plugin est constitué de 4 boutons radio permettant de sélectionner l'action à réaliser. Les possibilités sont l'enregistrement de toute une ontologie, l'enregistrement d'une partie d'ontologie, la suppression d'une ontologie, ou l'export de l'ontologie de type de données BioMoby vers une ontologie OWL. Les champs à remplir par l'utilisateur varient suivant l'action sélectionnée.

La deuxième partie de ce panel regroupe les différents champs à remplir par l'utilisateur. Ces champs permettent à l'utilisateur de rentrer tous les paramètres nécessaires à l'action souhaitée. Les champs présents sont les suivants :

- le nom de l'annuaire central (remplit automatiquement l'espace de nom et le point d'entrée)
- le point d'entrée de l'annuaire central ;
- l'espace de nom de l'annuaire central ;
- la racine de l'ontologie Biomoby ;
- la racine de l'ontologie OWL ;
- le chemin d'accès au dossier de cache.

Les différents champs sont détaillés ci-dessous.

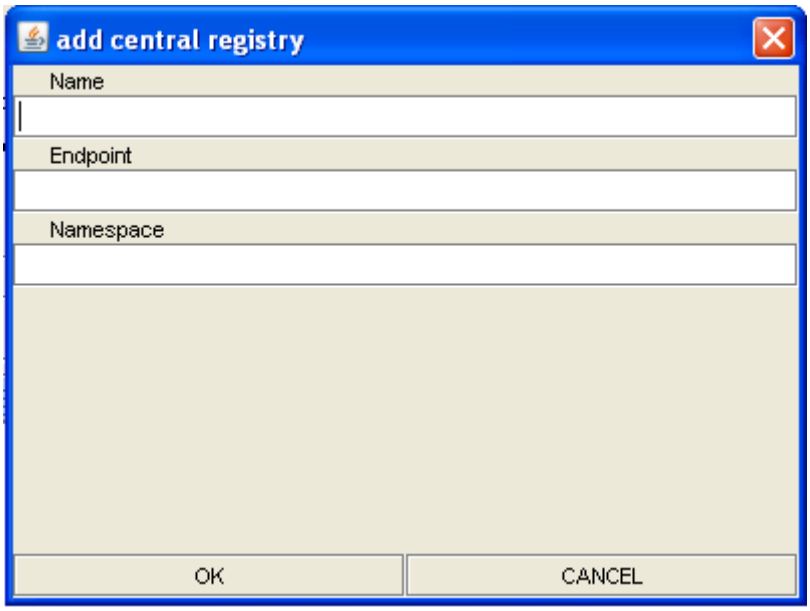
a) Les informations sur l'annuaire central Biomoby

Les informations sur l'annuaire central sont réparties sur 3 champs visibles par l'utilisateur. Ces 3 champs sont le nom de l'annuaire central, son espace de nom et son point d'entrée. En réalité l'utilisateur manipulera surtout le champ correspondant au nom de l'annuaire central. En effet, en sélectionnant un nom d'annuaire central, les autres champs seront remplis automatiquement. Ces informations sont stockées dans un fichier qui nous permet de récupérer les informations sur chaque annuaire central. Les informations sur chaque annuaire sont contenues sur 4 lignes. La première est une ligne contenant 10 étoiles. Cette ligne permet de marquer la séparation entre 2 annuaires et permet une meilleure lisibilité du fichier. La deuxième ligne comprend le nom de l'annuaire central. La 3ème ligne correspond au point d'entrée et la 4ème à l'espace de nom de l'annuaire.

```
**********
Calgary
http://moby.ucalgary.ca/moby/MOBY-Central.pl
http://moby.ucalgary.ca/MOBY/Central
**********
IRRI
http://cropwiki.irri.org/cgi-bin/MOBY-Central.pl
http://cropwiki.irri.org/MOBY/Central
```

Format du fichier contenant les informations de chaque annuaire

Lors de la première utilisation du plugin, le fichier est créé automatiquement et rempli avec les annuaires centraux d'enregistrement Biomoby les plus utilisés (Calgary, IRRI et INAB). Nous avons ensuite ajouté un bouton qui permet d'ouvrir une fenêtre. Cette fenêtre contient 3 champs qui permettent d'ajouter un annuaire central au fichier.



Fenêtre d'ajout d'un annuaire central

Si un utilisateur ne souhaite pas créer un nouvel annuaire, il peut également rentrer manuellement le point d'entrée et l'espace de nom correspondant.

b) Le préfixe de l'ontologie

Nous avons décidé d'ajouter la possibilité d'ajouter un préfixe à l'ontologie à enregistrer. Cette action est fortement recommandée. En effet, un annuaire central Biomoby est public. De nombreuses personnes peuvent donc ajouter leur ontologie. Cependant, deux types de données ne peuvent pas avoir le même nom. Dans le cas où nous souhaitons ajouter un type de données dont le nom est déjà présent sur l'annuaire deux cas sont possibles. Si le type de données présent possède des descendants ou s'il est utilisé par un Web Service, l'enregistrement de l'ontologie sera annulé et un message d'erreur sera renvoyé. Dans le cas ou le type de données déjà présent n'est pas encore utilisé dans Biomoby, celui-ci sera écrasé par le nouveau type de données enregistré. Afin de ne pas se voir écraser un type de données

ou de ne pas écraser un type de données existant, il est donc conseillé d'ajouter un préfixe au nom de chaque type de données que l'on souhaite enregistrer (si cela n'a pas été réalisé lors de la création de l'ontologie). C'est dans ce but que nous avons ajouté le champ préfixe. Le contenu de ce champ sera automatiquement ajouté devant chaque type de données enregistré.

c) La racine Biomoby

Ce champ correspond à l'endroit où nous souhaitons enregistrer notre ontologie sous Biomoby. Cela permet de ne pas être obligé d'ajouter une ontologie directement comme descendant de la racine de l'ontologie Biomoby. En effet, il est possible de vouloir rajouter une partie d'ontologie à une ontologie déjà enregistrée. Dans ce cas il suffit de sélectionné le type de données à partir duquel nous souhaitons enregistrer notre partie d'ontologie. La racine de l'ontologie de types de données Biomoby est un type de données qui s'appel « Object ». Si nous souhaitons enregistrer une ontologie directement à la racine de l'ontologie de types de données Biomoby la racine Biomoby à sélectionner sera donc « Object ».

Dans le cas ou nous souhaitons supprimer une ontologie, il suffit de choisir le type de données à supprimer le plus proche de la racine Biomoby. Ce type de données et tous ces descendants seront supprimés.

d) La racine OWL

Le champ permettant la sélection de la racine OWL n'est disponible que dans le cas où nous souhaitons enregistrer une partie d'ontologie. Cette racine OWL correspond à la classe de l'ontologie OWL à partir de laquelle nous souhaitons enregistrer tous les descendants.

Dans le cas où nous souhaitons ajouter une partie d'ontologie, ce champ peut être renseigné automatiquement en cliquant sur un nœud de l'arborescence de l'ontologie OWL. Cependant, il est également possible de le saisir manuellement.

e) Le chemin d'accès au dossier de cache

Le dernier champ de saisie permet de renseigner le chemin d'accès au dossier de cache. Ce dossier permet de stocker les informations récupérées dans un annuaire central Biomoby et ainsi d'accélérer grandement le trafic d'informations. Lors de la première utilisation du plugin il est recommandé de sélectionner un dossier cohérent. En effet, lors de la première utilisation les informations spécifiques à l'annuaire sélectionné sont récupérées dans ce dossier. Le fait de changer de dossier ultérieurement impliquera de réappliquer cette étape. Il s'agit d'une étape longue. Un chemin de cache standard (c:\cacheDir) est utilisé lors de la première utilisation du plugin. Ce chemin standard est stocké dans un fichier créé automatiquement lors de la première utilisation (le même fichier que pour les informations sur chaque annuaire). Il est possible de changer le chemin du dossier de cache via un bouton.

Annexe B : Détails des sorties obtenues lors des différentes étapes de transformation d'un plus court chemin en requête SPARQL

Dans cette partie nous allons présenter les sorties de chaque méthode permettant de transformer un plus court chemin sous condition vers une requête SPARQL. Ces sorties sont disponibles lors de la modification d'un booléen DEBUG. Les fichiers de sortie présentés ci-dessous sont issus de l'exemple utilisé dans la partie *création de requête* du chapitre *création automatique de requête SPARQL*.

1. Plus court chemin sous condition

sujet objet longueur typeDeNoeud typeDeRelation

null http://medoc.cirad.fr#phenotypingstudy_id_study 0.0 2 -1
null http://medoc.cirad.fr#phenotypingstudy_id_phenotypingstudy 0.0 3 -1
http://medoc.cirad.fr#phenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy 0.0 4 -1
http://medoc.cirad.fr#phenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy 0.98 1 2
null http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm 0.98 3 -1
null http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy 0.98 3 -1
http://medoc.cirad.fr#germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm 0.98 2 -1
http://medoc.cirad.fr#germplasmphenotypingstudy http://medoc.cirad.fr#germplasm_id_germplasm 0.98 5 -1
http://medoc.cirad.fr#germplasmphenotypingstudy http://medoc.cirad.fr#germplasm 2.02 1 0
null http://medoc.cirad.fr#genotypingstudy_id_study 0.0 2 -1
null http://medoc.cirad.fr#genotypingstudy_id_genotypingstudy 0.0 3 -1
http://medoc.cirad.fr#genotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy_id_genotypingstudy 0.0 4 -1
http://medoc.cirad.fr#genotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy 0.98 1 2
null http://medoc.cirad.fr#dnasamplegenotypingstudy_id_dnasample 0.98 3 -1
null http://medoc.cirad.fr#dnasamplegenotypingstudy_id_genotypingstudy 0.98 3 -1
http://medoc.cirad.fr#dnasamplegenotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy_id_dnasample 0.98 2 -1
http://medoc.cirad.fr#dnasamplegenotypingstudy http://medoc.cirad.fr#dnasample_id_dnasample 0.98 5 -1
http://medoc.cirad.fr#dnasamplegenotypingstudy http://medoc.cirad.fr#dnasample 1.98 1 0
null http://medoc.cirad.fr#dnasample_id_dnasample 1.98 3 -1
http://medoc.cirad.fr#dnasample http://medoc.cirad.fr#dnasample_id_germplasm 1.98 2 -1
http://medoc.cirad.fr#dnasample http://medoc.cirad.fr#germplasm_id_germplasm 1.98 5 -1
http://medoc.cirad.fr#dnasample http://medoc.cirad.fr#germplasm 3.02 1 0
null
null http://gcpdomainmodel.org/GCPDM#GCP_Study 0.0 0 -1
null http://medoc.cirad.fr#study_name 0.0 2 -1
null http://medoc.cirad.fr#study_id_study 0.0 3 -1
http://medoc.cirad.fr#study http://medoc.cirad.fr#phenotypingstudy_id_study 0.0 4 -1
http://medoc.cirad.fr#study http://medoc.cirad.fr#phenotypingstudy 1.0 1 1
null http://medoc.cirad.fr#phenotypingstudy_id_phenotypingstudy 1.0 3 -1
http://medoc.cirad.fr#phenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy 1.0 4 -1
http://medoc.cirad.fr#phenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy 1.98 1 2
null http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm 1.98 3 -1
null http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy 1.98 3 -1
http://medoc.cirad.fr#germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm 1.98 2 -1
http://medoc.cirad.fr#germplasmphenotypingstudy http://medoc.cirad.fr#germplasm_id_germplasm 1.98 5 -1
http://medoc.cirad.fr#germplasmphenotypingstudy http://medoc.cirad.fr#germplasm 3.02 1 0

sujet : table sur laquelle est pointé l'algorithme. Si sa valeur est bull, cela signifie que l'algorithme pointe sur une colonne.

objet : nœud (table ou colonne) pour lequel l'algorithme est actuellement en train de chercher des informations.

longueur : longueur du chemin à parcourir pour passer du nœud de départ au nœud objet. Cette valeur est égale à zéro si le nœud objet n'est pas une table.

typeDeNoeud : défini si le nœud objet est une table, une clé primaire, une clé étrangère, une clé étrangère inverse, le nœud de sortie, le nœud d'entrée ou une colonne ne rentrant dans aucune des conditions précédentes.

typeDeRelation : défini le type de relation permettant de passer d'une table à une autre. Sa valeur vaut -1 si le nœud objet est une colonne. Permet de différencier une relation d'héritage, une table d'association, une table d'association sans attributs ou une association d'aucun des types précédents.

2. Plus court chemin vers triplets

sujet prédicat objet

fin de bloc
?phenotypingstudy_id_phenotypingstudy http://medoc.cirad.fr#phenotypingstudy_id_study ?phenotypingstudy_id_study.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy ?phenotypingstudy_id_phenotypingstudy.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm ?germplasmphenotypingstudy_id_germplasm.
?germplasmphenotypingstudy_id_germplasm http://gcpdomainmodel.org/GCPDM#GCP_Germplasm ?germplasm_name.
UNION
?genotypingstudy_id_genotypingstudy http://medoc.cirad.fr#genotypingstudy_id_study ?genotypingstudy_id_study.
?key_dnasamplegenotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy_id_genotypingstudy ?genotypingstudy_id_genotypingstudy.
?key_dnasamplegenotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy_id_dnasample ?dnasamplegenotypingstudy_id_dnasample.
?dnasamplegenotypingstudy_id_dnasample http://medoc.cirad.fr#dnasample_id_germplasm ?dnasample_id_germplasm.
?dnasample_id_germplasm http://gcpdomainmodel.org/GCPDM#GCP_Germplasm ?germplasm_name.
fin de bloc
?study_id_study http://medoc.cirad.fr#study_name ?study_name.
?phenotypingstudy_id_phenotypingstudy http://medoc.cirad.fr#phenotypingstudy_id_study ?study_id_study.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy ?phenotypingstudy_id_phenotypingstudy.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm ?germplasmphenotypingstudy_id_germplasm.
?key_germplasm http://medoc.cirad.fr#germplasm_id_germplasm ?germplasmphenotypingstudy_id_germplasm.

fin de bloc: permet de savoir que tous les triplets présents avant cette ligne et après la dernière ligne fin de bloc parcourue sont des blocs de même niveau.

UNION : signifie que le bloc de triplets précédents cette ligne et celui suivant cette ligne seront de même niveau. Les données renvoyées par la requête finale correspondront à l'union de ces 2 blocs.

3. Gestion des blocs

Chaque bloc est composé d'un niveau, d'un type, du premier prédicat et de l'ensemble des triplets correspondants. Chaque bloc est séparé par une ligne vide.

niveau: 2
type: 0
premier prédicat: null
triplets:
null

niveau: 1
type: 1
premier prédicat: http://medoc.cirad.fr#phenotypingstudy_id_study
triplets:

?phenotypingstudy_id_phenotypingstudy http://medoc.cirad.fr#phenotypingstudy_id_study ?phenotypingstudy_id_study.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy ?phenotypingstudy_id_phenotypingstudy.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm ?germplasmphenotypingstudy_id_germplasm.
?germplasmphenotypingstudy_id_germplasm http://gcpdomainmodel.org/GCPDM#GCP_Germplasm ?germplasm_name.

niveau: 1
type: 2
premier prédicat: http://medoc.cirad.fr#genotypingstudy_id_study
triplets:
?genotypingstudy_id_genotypingstudy http://medoc.cirad.fr#genotypingstudy_id_study ?genotypingstudy_id_study.
?key_dnasamplegenotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy_id_genotypingstudy ?genotypingstudy_id_genotypingstudy.
?key_dnasamplegenotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy_id_dnasample ?dnasamplegenotypingstudy_id_dnasample.
?dnasamplegenotypingstudy_id_dnasample http://medoc.cirad.fr#dnasample_id_germplasm ?dnasample_id_germplasm.
?dnasample_id_germplasm http://gcpdomainmodel.org/GCPDM#GCP_Germplasm ?germplasm_name.

niveau: 0
type: 1
premier prédicat: http://medoc.cirad.fr#study_name
triplets:
?study_id_study http://medoc.cirad.fr#study_name ?study_name.
?phenotypingstudy_id_phenotypingstudy http://medoc.cirad.fr#phenotypingstudy_id_study ?study_id_study.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy ?phenotypingstudy_id_phenotypingstudy.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm ?germplasmphenotypingstudy_id_germplasm.
?key_germplasm http://medoc.cirad.fr#germplasm_id_germplasm ?germplasmphenotypingstudy_id_germplasm.

5. Gestion de la requête

Consiste à prendre en compte les blocs pour créer la structure finale de la requête.

```
SELECT DISTINCT ?study_name ?germplasm_name WHERE {
FILTER regex(?study_name,"^inputString$").
?study_id_study http://gcpdomainmodel.org/GCPDM#GCP_Study ?study_name.
{
?phenotypingstudy_id_phenotypingstudy http://medoc.cirad.fr#phenotypingstudy_id_study ?phenotypingstudy_id_study.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_phenotypingstudy
?phenotypingstudy_id_phenotypingstudy.
?key_germplasmphenotypingstudy http://medoc.cirad.fr#germplasmphenotypingstudy_id_germplasm
?germplasmphenotypingstudy_id_germplasm.
?germplasmphenotypingstudy_id_germplasm http://gcpdomainmodel.org/GCPDM#GCP_Germplasm ?germplasm_name.
}
UNION
{
?genotypingstudy_id_genotypingstudy http://medoc.cirad.fr#genotypingstudy_id_study ?genotypingstudy_id_study.
?key_dnasamplegenotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy_id_genotypingstudy
?genotypingstudy_id_genotypingstudy.
?key_dnasamplegenotypingstudy http://medoc.cirad.fr#dnasamplegenotypingstudy_id_dnasample
?dnasamplegenotypingstudy_id_dnasample.
?dnasamplegenotypingstudy_id_dnasample http://medoc.cirad.fr#dnasample_id_germplasm ?dnasample_id_germplasm.
?dnasample_id_germplasm http://gcpdomainmodel.org/GCPDM#GCP_Germplasm ?germplasm_name.
}
}
```

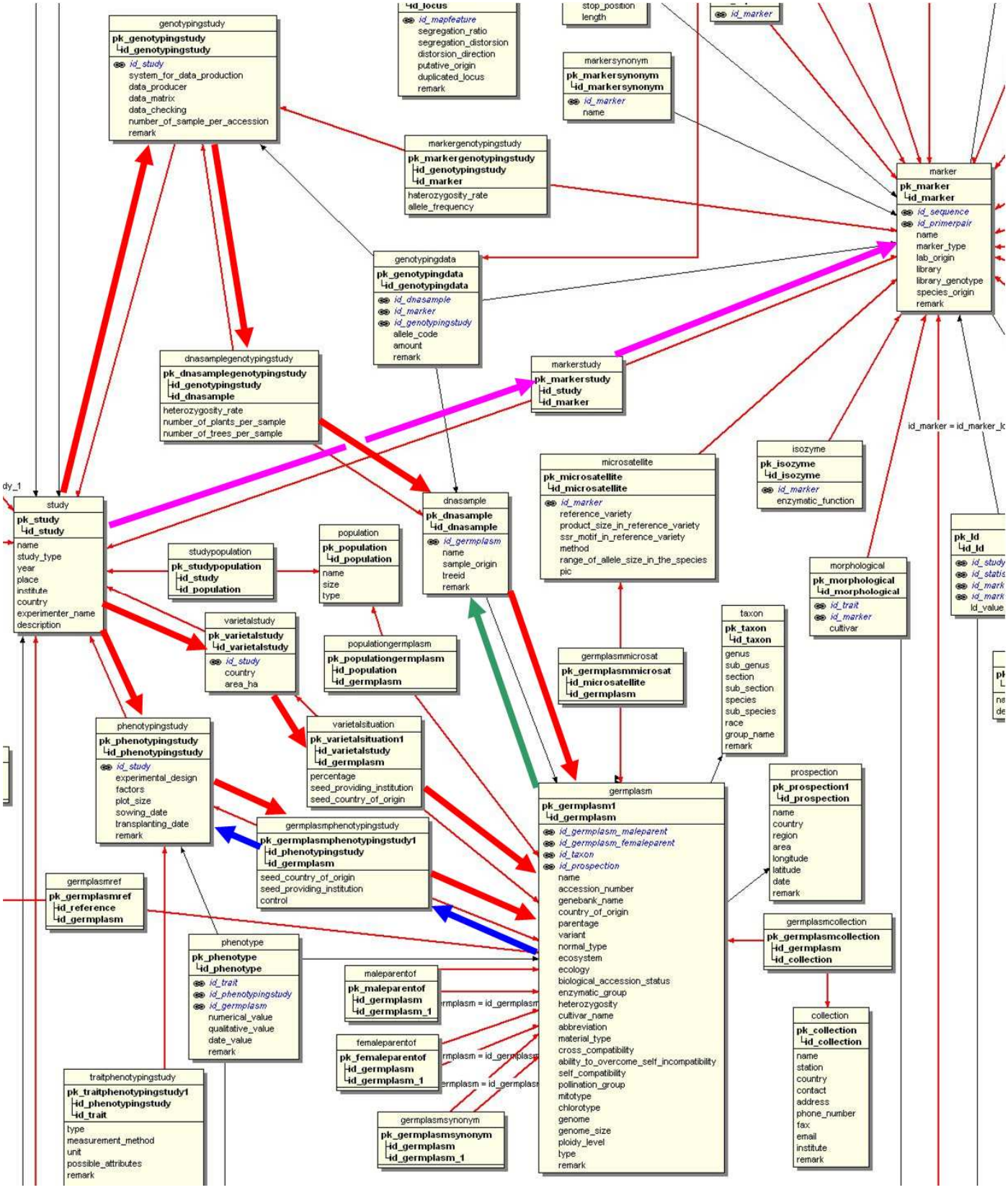
6. Requête finale

Requête finale pouvant être ajoutée à un Service Web Sémantique créé par la plateforme BioSemantic. Cette méthode permet de remplacer les URIs des ressources par des préfixes et rajoute au début de la requête la description des préfixes utilisés.

```
PREFIX vocab: <jdbc:mysql://medoc.cirad.fr/TROPGENE_RICE/vocab#>
PREFIX gcpdm: <http://gcpdomainmodel.org/GCPDM#>
SELECT DISTINCT ?study_name ?germplasm_name WHERE {
  FILTER regex(?study_name,"^inputString$").
  ?study_id_study gcpdm:GCP_Study ?study_name.
  {
    ?phenotypingstudy_id_phenotypingstudy vocab:phenotypingstudy_id_study ?study_id_study.
    ?key_germplasmphenotypingstudy vocab:germplasmphenotypingstudy_id_phenotypingstudy
?phenotypingstudy_id_phenotypingstudy.
    ?key_germplasmphenotypingstudy vocab:germplasmphenotypingstudy_id_germplasm
?germplasmphenotypingstudy_id_germplasm.
    ?germplasmphenotypingstudy_id_germplasm gcpdm:GCP_Germplasm ?germplasm_name.
  }
  UNION
  {
    ?genotypingstudy_id_genotypingstudy vocab:genotypingstudy_id_study ?study_id_study.
    ?key_dnasamplegenotypingstudy vocab:dnasamplegenotypingstudy_id_genotypingstudy
?genotypingstudy_id_genotypingstudy.
    ?key_dnasamplegenotypingstudy vocab:dnasamplegenotypingstudy_id_dnasample
?dnasamplegenotypingstudy_id_dnasample.
    ?dnasamplegenotypingstudy_id_dnasample vocab:dnasample_id_germplasm ?dnasample_id_germplasm.
    ?dnasample_id_germplasm gcpdm:GCP_Germplasm ?germplasm_name.
  }
}
```

Annexe C : Requêtes utilisées pour le benchmark de la création de requêtes SPARQL

Cette figure représente le modèle logique de Tropgene. Les flèches larges présentes les chemins utilisés pour créer manuellement des requêtes SQL. La requête 1 est présentée par des flèches rouges, la requête 2 par des flèches roses, la requête 3 par des flèches bleues et la requête 4 par une flèche verte. La requête 5 peut être représentée par les flèches de la requête 1 mais orientées dans le sens opposé.



Nous allons maintenant présenter les différentes requêtes dans leur langage de requête respectif. Pour les 4 premières requêtes, les éléments parcourus dans le schéma relationnel sont les mêmes pour les requêtes SQL créées manuellement et les requête SPARQL créées automatiquement. Pour la requête 5 la requête SPARQL ne comporte qu'une partie de la requête SQL initiale. Cela s'explique par une erreur lors de l'implémentation empêchant la détection de relations d'héritage parcourues dans le sens relation spécialisée vers relation généraliste. Ce bug va être modifié rapidement.

Requête 1 : Flèches rouges

SQL

```
SELECT DISTINCT S.name, G.name
FROM study S, genotypingstudy GS, dnasamplegenotypingstudy DGS, dnasample D, germplasm G
WHERE
   S.id_study = GS.id_study AND GS.id_genotypingstudy = DGS.id_genotypingstudy AND
   DGS.id_dnasample = D.id_dnasample AND D.id_germplasm = G.id_germplasm
UNION
SELECT S.name, G.name
FROM study S, varietalstudy VST, varietalsituation VSI, germplasm G
WHERE
   S.id_study = VST.id_study AND VST.id_varietalstudy = VSI.id_varietalstudy AND
   VSI.id_germplasm = G.id_germplasm
UNION
SELECT S.name, G.name
FROM study S, phenotypingstudy PS, germplasmphenotypingstudy GPS, germplasm G
WHERE
   S.id_study = PS.id_study AND PS.id_phenotypingstudy = GPS.id_phenotypingstudy AND
   GPS.id_germplasm = G.id_germplasm
```

SPARQL

```
SELECT DISTINCT ?study_name ?germplasm_name WHERE {
FILTER regex(?study_name,"^.*$").
?study_id_study gcpdm:GCP_Study ?study_name.
{
?varietalstudy_id_varietalstudy vocab:varietalstudy_id_study ?study_id_study.
?key_varietalsituation vocab:varietalsituation_id_varietalstudy ?varietalstudy_id_varietalstudy.
?key_varietalsituation vocab:varietalsituation_id_germplasm ?varietalsituation_id_germplasm.
?varietalsituation_id_germplasm gcpdm:GCP_Germplasm ?germplasm_name.
}
UNION
{
?genotypingstudy_id_genotypingstudy vocab:genotypingstudy_id_study ?study_id_study.
?key_dnasamplegenotypingstudy vocab:dnasamplegenotypingstudy_id_genotypingstudy ?genotypingstudy_id_genotypingstudy.
?key_dnasamplegenotypingstudy vocab:dnasamplegenotypingstudy_id_dnasample ?dnasamplegenotypingstudy_id_dnasample.
?dnasamplegenotypingstudy_id_dnasample vocab:dnasample_id_germplasm ?dnasample_id_germplasm.
?dnasample_id_germplasm gcpdm:GCP_Germplasm ?germplasm_name.
}
UNION
{
?phenotypingstudy_id_phenotypingstudy vocab:phenotypingstudy_id_study ?study_id_study.
?key_germplasmphenotypingstudy vocab:germplasmphenotypingstudy_id_phenotypingstudy
?phenotypingstudy_id_phenotypingstudy.
?key_germplasmphenotypingstudy vocab:germplasmphenotypingstudy_id_germplasm ?germplasmphenotypingstudy_id_germplasm.
?germplasmphenotypingstudy_id_germplasm gcpdm:GCP_Germplasm ?germplasm_name.
}
}
```

Requête 2 : Flèches roses

SQL

```
SELECT DISTINCT S.name, M.name
FROM study S, markerstudy MS, marker M
WHERE S.id_study = MS.id_study AND MS.id_marker = M.id_marker
```

SPARQL

```
SELECT DISTINCT ?marker_name ?study_name WHERE {
FILTER regex(?marker_name,"^.*$").
?marker_id_marker gcpdm:GCP_GenomicFeatureDetector ?marker_name.
?marker_id_marker vocab:markerstudy ?study_id_study.
?study_id_study gcpdm:GCP_Study ?study_name.
}
```

Requête 3 : Flèches bleues

SQL

```
SELECT DISTINCT G.name, PS.factors
FROM germplasm G, germplasmphenotypingstudy GPS, phenotypingstudy PS
WHERE
   G.id_germplasm = GPS.id_germplasm AND
   GPS.id_phenotypingstudy = PS. Id_pehnotypingstudy
```

SPARQL

```
SELECT DISTINCT ?germplasm_name ?phenotypingstudy_factors WHERE {
FILTER regex(?germplasm_name,"^inputString$").
?germplasm_id_germplasm gcpdm:GCP_Germplasm ?germplasm_name.
?key_germplasmphenotypingstudy vocab:germplasmphenotypingstudy_id_germplasm ?germplasm_id_germplasm.
?key_germplasmphenotypingstudy vocab:germplasmphenotypingstudy_id_phenotypingstudy
?germplasmphenotypingstudy_id_phenotypingstudy.
?germplasmphenotypingstudy_id_phenotypingstudy gcpdm:GCP_Factor ?phenotypingstudy_factors.
}
```

Requête 4 : Flèche verte

SQL

```
SELECT G.name, D.name
FROM germplasm G, dnasample D
WHERE G.id_germplasm = D.id_germplasm
```

SPARQL

```
SELECT DISTINCT ?germplasm_name ?dnasample_name WHERE {
FILTER regex(?germplasm_name,"^.*$").
?germplasm_id_germplasm gcpdm:GCP_Germplasm ?germplasm_name.
?dnasample_id_germplasm vocab:dnasample_id_germplasm ?germplasm_id_germplasm.
?dnasample_id_germplasm gcpdm:GCP_SimpleIdentifier ?dnasample_name.
}
```

Requête 5 : Requête correspondent aux flèches rouges orientées dans le sens oppose

SQL

```
SELECT DISTINCT G.name, S.name
FROM germplasm G, dnasample D, dnasamplegenotypingstudy DGS, genotypingstudy GS, study S
WHERE
   D.id_germplasm = G.id_germplasm AND DGS.id_dnasample = D.id_dnasample AND
UNION
SELECT G.name, S.name
FROM germplasm G, varietalsituation VSI, varietalstudy VST, study S
WHERE
   VSI.id_germplasm = G.id_germplasm AND VST.id_varietalstudy = VSI.id_varietalstudy AND
   S.id_study = VST.id_study
UNION
SELECT G.name, S.name
FROM germplasm G, germplasmphenotypingstudy GPS, phenotypingstudy PS, study S
WHERE
   GPS.id_germplasm = G.id_germplasm AND PS.id_phenotypingstudy = GPS.id_phenotypingstudy AND
   S.id_study = PS.id_study
```

SPARQL

```
SELECT DISTINCT ?germplasm_name ?study_name WHERE {
FILTER regex(?germplasm_name,"^.*$").
?germplasm_id_germplasm gcpdm:GCP_Germplasm ?germplasm_name.
?key_germplasmphenotypingstudy vocab:germplasmphenotypingstudy_id_germplasm
?germplasm_id_germplasm.
?key_germplasmphenotypingstudy vocab:germplasmphenotypingstudy_id_phenotypingstudy
?germplasmphenotypingstudy_id_phenotypingstudy.
?germplasmphenotypingstudy_id_phenotypingstudy vocab:phenotypingstudy_id_study
?phenotypingstudy_id_study.
?phenotypingstudy_id_study gcpdm:GCP_Study ?study_name.
}
```

Annexe D : Détail des étapes d'enregistrement automatique d'un Service Web Sémantique

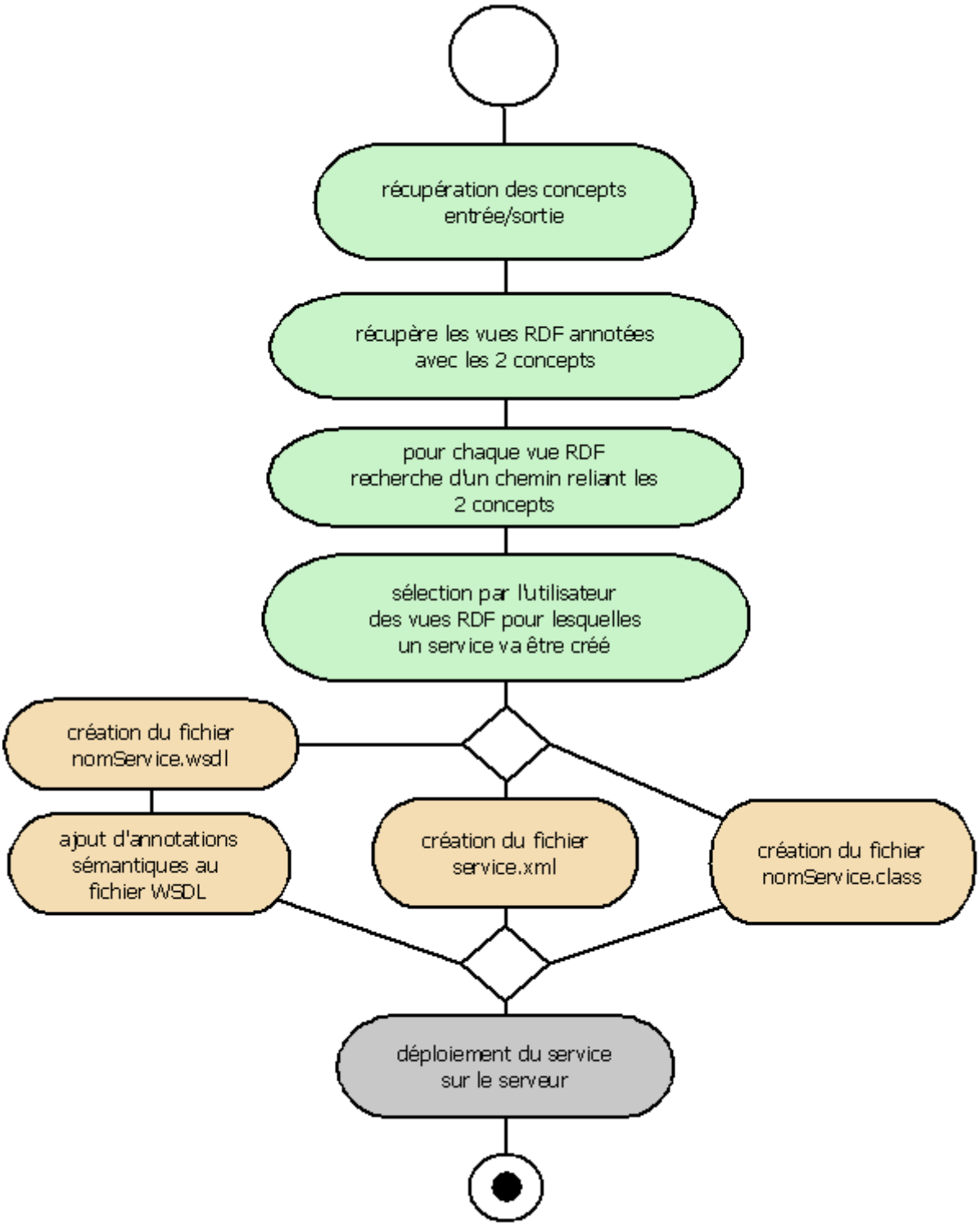


Diagramme d'activité de l'enregistrement d'un Service Web Sémantique via l'application Web BioSemantic

Recherche des vues RDF

La création de nos SWS est basée sur l'annotation des vues RDF hétérogènes avec des concepts ontologiques identiques. Nous voulons que pour un couple de concepts ontologiques

donnés, toutes les vues annotées avec ces concepts permettent la création d'un SWS. Pour cela, il faut détecter, parmi les vues RDF présentent dans le dossier RDF, celles qui sont annotées avec les deux concepts. Chaque vue RDF est parcourue, et les chemins d'accès des vues annotées avec les deux concepts sont conservées.

Lorsqu'une vue RDF est annotée avec les 2 concepts et qu'un chemin permet de passer d'un concept à un autre dans le graphe, une requête SPARQL est créée automatiquement.

Création des éléments du Service Web sémantique

La création d'un SWS nécessite plusieurs étapes et notamment la création de plusieurs fichiers. Ces fichiers permettront par la suite de déployer automatiquement les Services sur un serveur Web.

Création de la classe d'implémentation

Elle représente l'implémentation du Service Web. Dans notre cas cette classe est développée en Java sous Eclipse. C'est dans cette classe que seront implémentées les méthodes du Service. Dans notre cas, quel que soit le SWS créé, la classe d'implémentation comportera une méthode ayant un paramètre d'entrée et un paramètre de sortie. Nous avons donc décidé de créer une classe Java générique (Figure 73).

```
public class GenericServiceImpl
{
  private static final Logger LOG = Logger.getLogger(GenericServiceImpl.class);
  static public final ThreadLocal serviceName = new ThreadLocal();
  public Vector<String> method ( String input )
  {
String sServiceName = (String) serviceName.get();
String[] serviceContext =
fr.cirad.test.servlet.CustomAxisServlet.getServiceN3params(sServiceName);
Vector<String> output=new Vector<String>();
if (serviceContext != null){
String inputConcept="";
String inputColumn="";
String outputConcept="";
String outputColumn="";
String mappingFile=CustomAxisServlet.getN3RepositoryPath() + serviceContext[0];
ModelD2RQ m = new ModelD2RQ(mappingFile);
String sparql=serviceContext[1];
for (int i=2; i<serviceContext.length; i++){
String line=serviceContext[i];
//select input and output columns
if(line.startsWith("SELECT")){
String[] lineElements=line.split(" ");
//if WS contains one input and one output
if(lineElements.length==6){
inputColumn=lineElements[2];
```

```
outputColumn=lineElements[3];
}
//if WS only contains one output
if(lineElements.length==5){
outputColumn=lineElements[2];
}
}
else if(line.contains("inputString")){
line=line.replaceAll("inputString", input.toString());
}
//select input and output concepts
else if(line.contains(inputColumn)){
inputConcept=line.split(" ")[1];
}
else if(line.contains(outputColumn)){
outputConcept=line.split(" ")[1];
}
sparql += "\n" + line;
}
ResultSet rs1 = QueryExecutionFactory.create(sparql, m).execSelect();
while (rs1.hasNext()) {
    QuerySolution row = rs1.nextSolution();
    output.add(

"<"+inputConcept+"/>"+row.getLiteral(inputColumn.substring(1)).getString()+"<"+outp
utConcept+"/>"+row.getLiteral(outputColumn.substring(1)).getString());
}
}
return output;
  }
}
```

Figure 73 : classe d'implémentation générique du Service Web

Certains paramètres de cette classe n'auront pas la même valeur selon le SWS que l'on souhaite créer. Pour cela, nous avons décidé de stocker ces paramètres dans un autre fichier et de les récupérer à la volée.

Création du fichier services.xml

Il s'agit du fichier contenant tous les paramètres nécessaires à l'utilisation du SWS. Initialement ce fichier contient 3 paramètres indispensables :

le nom du service web : c'est ce nom qui est utilisé pour appeler le Service.

La classe du Service : correspond un nom de la classe Java de l'implémentation du Service

Utilisation du WSDL original : Ce paramètre peut prendre la valeur oui ou non. S'il est défini à oui, alors un fichier WSDL sera créé automatiquement. S'il est défini à non, il faut créer son propre fichier WSDL qui sera pris en compte pour décrire le Service Web.

Dans notre cas l'utilisation de ces seuls paramètres n'était pas suffisante. Nous avons donc créé deux paramètres supplémentaires qui nous permettront d'utiliser notre classe générique Java. Ces paramètres sont :

chemin du fichier N3 : paramètre qui défini le chemin vers la vue RDF utilisée par le Service Web.

Requête : paramètre qui regroupe toutes les lignes de la requête créée automatiquement.

Un exemple de fichier services.xml est présent dans la page suivante.

<service name="get_medoc_TROPGENE_COTTON_GCP_Study_accession">
<messageReceivers>
<messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-only"
class="org.apache.axis2.rpc.receivers.RPCInOnlyMessageReceiver" />
<messageReceiver mep="http://www.w3.org/2004/08/wsdl/in-out" class="org.apache.axis2.rpc.receivers.RPCMessageReceiver"
/>
</messageReceivers>
<parameter name="ServiceClass" locked="false">fr.cirad.test.service.impl.GenericServiceImpl</parameter>
<parameter name="useOriginalwsdl">true</parameter>
<parameter name="n3FilePath"
locked="false">D:\workspace2\.metadata\.plugins\org.eclipse.wst.server.core\tmp1\wtpwebapps\BioSemantic_webapp\WEB-
INF\repository\mapping-MEDOC-TROPGENE_COTTON.n3</parameter>
<parameter name="query_0" locked="false">PREFIX vocab:
<jdbc:mysql://medoc.cirad.fr/TROPGENE_COTTON/vocab#></parameter>
<parameter name="query_1" locked="false">PREFIX gcpdm: <http://gcpdomainmodel.org/GCPDM#></parameter>
<parameter name="query_2" locked="false">SELECT DISTINCT ?study_name ?sequence_accession_number WHERE {</parameter>
<parameter name="query_3" locked="false">FILTER regex(?study_name,"^inputString$").</parameter>
<parameter name="query_4" locked="false">?study_id_study gcpdm:GCP_Study ?study_name.</parameter>
<parameter name="query_5" locked="false">?marker_id_marker vocab:markerstudy ?study_id_study.</parameter>
<parameter name="query_6" locked="false">?sequence_id_marker vocab:sequence_id_marker ?marker_id_marker.</parameter>
<parameter name="query_7" locked="false">?marker_id_marker gcpdm:accession ?sequence_accession_number.</parameter>
<parameter name="query_8" locked="false">}</parameter>
</service>

Description des paramètres

ServiceClass : classe Java contenant l'implémentation du Service Web

useOriginalWSDL : permet de préciser si un fichier WSDL a été créé ou s'il doit être généré à la volée par Axis. La valeur true indique qu'il a été créé physiquement.

N3FilePath : chemin d'accès vers la vue RDF utilisée pour créer la requête du Service Web. Cette vue RDF sera utilisée pour interroger la base de données d'origine lors de l'appel du Service Web

query_N : avec N allant de 0 au nombre de lignes de la requête. Ce paramètre permet de stocker la requête.

Création du fichier de description

Les descriptions du Service sont présentes dans un fichier nommé fichier WSDL pour *Web Service Description Language*. Ce fichier WSDL est un fichier XML contenant une description de tout ce qui est nécessaire à l'appel d'un Service Web SOAP à savoir :

l'URL et l'espace de nom du Service

Le type de Service

La liste des méthodes disponibles

Les arguments de chaque méthode

Le type de données de chaque argument

La valeur de retour de chaque méthode et le type de données de chaque valeur de retour

Le fichier WSDL peut être géré de 2 manières différentes. Soit il est généré à la volée soit il est créé manuellement puis est stocké physiquement sur le serveur contenant le Service Web qu'il décrit. Dans notre cas nous avons décidé de créer ce fichier physiquement. Cela nous permet de le modifier pour ajouter des annotations sémantiques.

Annotation sémantique du fichier de description

Les 3 fichiers que nous venons de créer sont suffisants pour créer automatiquement un Service Web. Pour l'annotation sémantique des Services Web, nous avons décidé d'utiliser l'approche recommandée par le W3C, à savoir SAWSDL (Farrell & Lausen 2007).

Les annotations SAWSDL sont intégrées directement dans le fichier WSDL de description du SWS sous la forme de balises supplémentaires. Ces balises, détaillées dans la Figure 41, sont ajoutées automatiquement au fichier WSDL créé précédemment. Dans cette figure, l'annotation SAWSDL est présentée en gras. Dans cet exemple, elle permet d'annoter sémantiquement l'attribut d'entrée de la méthode contenue dans notre SWS à l'aide du concept ontologique correspondant.

```
<xs:element name="method">
  <xs:complexType>
    <xs:sequence>
      <xs:element minOccurs="0" name="input" nillable="true"  type="xs:string"
      sawsdl:modelReference="http://gcpdomainmodel.org/GCPDM#GCP_GenotypeStudy"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Figure 74 : Exemple d'annotation sémantique de l'entrée d'un Service Web à l'aide d'une balise SAWSDL

Déploiement automatique

Nos SWS sont déployés sur un serveur Web Tomcat[19] en utilisant le serveur SOAP Axis2[20]. Il s'agit d'une API open-source permettant de développer et déployer des Services Web SOAP. Elle permet également de les déployer automatiquement en créant les fichiers nécessaires tout en respectant une arborescence de fichier spécifique.

La phase de déploiement de nos SWS consiste donc en une simple étape de copie sur le serveur des fichiers décrits précédemment.

Enregistrement dans un annuaire de Services Web

Nous avons décidé d'utiliser un des annuaires principaux de Services Web bioinformatiques : BioCatalogue (Bhagat et al. 2010). Il s'agit d'un annuaire hébergé à l'EMBL-EBI permettant d'enregistrer tout type de Servicse Web spécifiques au domaine de la biologie. Il n'héberge pas les Service Web eux-même mais se charge d'offrir un moyen de les découvrir et de les annoter sémantiquement. BioCatalogue nous permettra d'enregistrer facilement nos SWS et d'utiliser leurs annotations sémantiques pour faciliter leurs découvertes par d'autres utilisateurs.

[19] http://tomcat.apache.org/
[20] http://axis.apache.org/axis2/java/core/

Annexe E : Création de bases de données relationnelles

La littérature nous a permis de déterminer que la majorité des données biologiques publiques disponibles sur le Web sont stockées dans des bases de données relationnelles. Le processus standard de création d'une base de données relationnelle est composé de trois étapes principales qui sont la création d'un schéma conceptuel, la création du schéma logique et la création du schéma physique (Ramakrishnan & Gehrke 2002).

Le modèle entité association

Habituellement, la phase de modélisation intervient après une étape préliminaire où les besoins d'interrogation et les données à stocker dans la future base de données sont analysés. Cette phase conceptuelle conduit à la création d'un schéma conceptuel décrivant les données qui seront stockées dans la base de données et les contraintes qu'elles nécessitent. Le modèle conceptuel le plus connu dans la conception de bases de données est le modèle entité association proposé par Peter Chen (P. P.-shan Chen 1976).

Les éléments principaux de ce modèle sont (Kiringa 2009) :

- entité : il s'agit d'un objet du monde réel, distinct des autres objets. Il est décrit à l'aide d'attributs,
- ensemble d'entités : il s'agit d'une collection d'entités similaires,
- attribut : il est associé à un ensemble d'entités. Toutes les entités appartenant à un même ensemble d'entités sont décrites à l'aide des mêmes attributs. Pour chaque ensemble d'entités, une clé est définie comme étant le plus petit ensemble d'attributs permettant d'identifier une entité de façon unique,
- association : permet d'associer n entités, n étant appelé l'arité de l'association. Une association peut également être décrite à l'aide d'attributs,
- ensemble d'associations : il s'agit d'une collection d'associations similaires,
- cardinalité : défini le nombre minimum et maximum d'entités d'un ensemble d'entités qui peuvent participer à une association pour un ensemble d'associations données,
- entité faible : entité dépendant de la présence d'une autre appelée entité identifiante. Elle peut être définie seulement si l'on considère la clé primaire de l'entité identifiante,
- ensemble d'entités faible : correspond à une collection d'entités faibles. Une entité faible est associée à une seule entité propriété identifiante et une propriété identifiante peut être associée à une multitude de propriétés faibles.

De nos jours, une version étendue de ce modèle est utilisée pour conceptualiser une base de données. Il s'agit du modèle appelé entité association étendu qui rajoute la possibilité de définir des relations d'héritage, d'union et d'agrégation.

Le modèle relationnel

Le modèle logique correspond à un modèle de représentation qui est situé entre le modèle entité association et le modèle physique, permettant de définir les détails de stockage de bas niveau de la base de données. Le modèle le plus utilisé est le modèle relationnel présenté par Edgar Codd (E. F. Codd 1970). Les éléments principaux de ce modèle sont :

- relations : qui peuvent être vu comme des tables,
- attributs : qui peuvent être visualisés comme les colonnes d'une table. Chaque attribut appartient à une seule relation,
- tuple : une instance d'une relation est appelée un tuple,
- clé primaire : le plus petit ensemble d'attributs permettant de définir un tuple est appelé clé candidate. S'il existe plusieurs clés candidates, l'une d'entre elles est définie comme la clé primaire, et est utilisée pour identifier un tuple de façon unique,
- clé étrangère : une clé étrangère pour une relation est formée d'un ou plusieurs attributs qui constituent une clé primaire dans une autre relation.

Le passage du modèle entité association au modèle relationnel est réalisé en suivant des algorithmes standards bien définis (Elmasri 2006). Le schéma relationnel obtenu n'est pas forcément optimal et nécessite des étapes de normalisation (E.F Codd 1971), permettant notamment de réduire la redondance des données stockées dans la base de données et ainsi éviter des problèmes lors d'opérations de mise à jour. Les règles de normalisation les plus utilisées sont appelées 1NF (première forme normale), 2NF, 3NF (E.F Codd 1971) et la forme normale Boyce-Codd (BCNF) (Edgar F. Codd 1974).

Il existe également des approches d'ingénierie inverse (K. H. Davis & Arora 1987; Comyn-Wattiau & Akoka 1996), qui consistent à essayer de passer du modèle logique ou physique vers un modèle conceptuel. Les règles de passage d'un modèle conceptuel vers un modèle logique ne sont pas toutes réversibles. Il n'est donc pas possible de récupérer toutes les informations du modèle conceptuel à partir d'un modèle relationnel. Les approches d'ingénierie inversent s'appuient donc sur des heuristiques permettant de récupérer un maximum d'informations du modèle conceptuel initial.

Le modèle physique

Le passage du modèle relationnel au modèle physique consiste à créer physiquement la base de données relationnelle dans un système de gestion de base de données relationnel (RDBMS) donné, en s'appuyant sur le modèle relationnel. Les RDBMS les plus utilisés sont MySQL, Oracle, et PostgreSQL.

SQL

Une grande partie du succès des bases de données relationnelles est due à SQL (Structured Query Language), le langage de requête standard des bases de données relationnelles. SQL est basé sur l'algèbre relationnel (E. F. Codd 1970). Toutes requêtes pouvant être représentées sous forme d'algèbre relationnel peuvent donc être représentées en SQL.

Résumé

Le partage de l'information sur le Web a permis aux biologistes d'accéder à de multiples sources de données souvent nécessaires pour répondre à des questions biologiques complexes. Cependant, l'accès indépendant à ces sources et leur forte évolutivité, limite le degré d'automatisation de la recherche d'information. Dans ce contexte des solutions intégrant le contenu de plusieurs sources de données ont vu le jour. Parmi les approches classiques d'intégration, la médiation est une approche virtuelle simulant une intégration physique des données par l'utilisation d'adaptateurs interrogeant directement les sources de données dans leurs emplacements d'origine. Cependant, la création de ces adaptateurs est consommatrice de temps.
Les travaux introduits dans ce mémoire proposent d'automatiser la création d'adaptateurs sémantiques, facilitant ainsi l'intégration et le partage de sources de données biologiques. Plus précisément, nous nous sommes intéressés au développement d'une méthodologie permettant d'automatiser au maximum l'intégration de bases de données relationnelles biologiques dans une plateforme d'intégration de type médiation. Notre approche se veut flexible, générique et automatisée. Pour cela nous nous appuyons sur des standards du Web Sémantique et des Services Web pour développer de manière automatisée des adaptateurs sémantiques de type Service Web Sémantique.

Abstract :

The sharing of data on the Web has allowed researchers to access multiple data sources in order to address complex biological questions. However, independent access to these sources and their high scalability, limit automatic information retrieval. In this context, several solutions that integrate data from multiple sources have emerged. Among the traditional integration approaches, mediation is a virtual integration which uses wrappers to gather data sources from their original locations. However, the creation of these wrappers is a time consuming task.
Our work proposes to automate the creation of semantic wrappers, thus facilitating the integration of biological data sources. More precisely, we focused on developing a methodology that automates the integration of relational databases in the context of mediation systems. Our approach is flexible, generic and automated. For this we used standards of the Semantic Web and Web Services to develop automated semantic wrappers as Semantic Web Services.

Résumé:

Les données et les sources biologiques sont hétérogènes (syntaxe et sémantique), de plus en plus nombreuses et fortement évolutives. L'agrégation de données ayant de telles spécificités est complexe et nécessite le développement de solutions d'intégration. La médiation est une approche virtuelle simulant une intégration physique des données par l'utilisation d'adaptateurs interrogeant directement les sources de données dans leurs emplacements d'origine. Cependant, la création de ces adaptateurs est consommatrice de temps.
Les travaux introduits dans ce mémoire proposent d'automatiser la création d'adaptateurs sémantiques, facilitant ainsi l'intégration et le partage de sources de données biologiques. Plus précisément, nous nous sommes intéressés au développement d'une méthodologie permettant d'automatiser au maximum l'intégration de bases de données relationnelles biologiques dans une plateforme d'intégration de type médiation. Notre approche se veut flexible, générique et automatisée. Pour cela nous nous appuyons sur des standards du Web Sémantique et des Services Web pour développer de manière automatisée des adaptateurs sémantiques de type Service Web Sémantique.

Abstract :

Biological data and data sources are heterogeneous (syntax and semantic), more and more numerous and highly scalable. Aggregating data with such specificities is complex and requires development of integration solutions. Mediation is an approach to simulating a virtual physical integration and sharing of data through the use of wrappers, which directly queried data sources in their original locations. However, wrapper creation is time consuming. Our work proposes to automate the creation of semantic wrappers, thus facilitating the integration of biological data sources. Specifically, we were interested in developing a methodology to automate the integration of relational databases in mediation like integration platform. Our approach is flexible, generic and automated. For this we used standards of the Semantic Web and Web Services to develop automated semantic wrappers as Semantic Web Services.